UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE GRADO

GRADO EN INGENIERIA INFORMÁTICA

DESARROLLO DE UN ENTORNO DE MONITORIZACIÓN CENTRALIZADA

TUTOR: DAVID EXPÓSITO SINGH
AUTOR: ALEJANDRO GARCÍA-CANTARERO ALAÑÓN

A mis padres y mi hermana por haber aguantado todos mis agobios.

A mis abuelos y a mi familia por haber confiado siempre en mí.

A Andrea L., Carolina, Alba, Álvaro, Andrea U., Teresa, Rocío y Robert por todo el apoyo.

A Sara y Raquel por ayudarme a compaginar mi vida músico-artística con el proyecto.

A Elena, Álvaro M., Álvaro R., Marta, Josemi, Paula, Juan y Chema por ser un apoyo a mi locura.

A Alfonso por enseñarme a no rendirme jamás,

A Patricia y a Zoraida por abrirme el mundo del Machine Learning,

Al Caballero Oscuro por tantas noches en vela.

Y a David, por guiarme por todo este proyecto y ayudarme en todo momento.

"Cerca trova /

Busca y hallarás"

Giorgio Vasari, "La Batalla Marciano en Val di Chiana".

Resumen

El rendimiento es un factor muy importante en la computación hoy en día. Numerosos sectores del mercado, así como investigadores, se centran más en buscar rendimiento que otra cualidad de los sistemas. Esto es debido al aumento del uso de los supercomputadores y clústeres, sistemas de computación de alto rendimiento que basa su potencia en ejecución en paralelo de gran cantidad de procesos o hilos. Sin embargo estos sistemas tienen unos recursos limitados, y en numerosas ocasiones no son suficientes para el objetivo por el que se utilizan.

Es por esto por lo que se busca el mayor rendimiento en estos sistemas, es decir, que los sistemas tengan mecanismos, u otras salidas, que permitan detectar y corregir la existencia de cuellos de botella en el rendimiento, usando al máximo todos los recursos de la máquina. Un ejemplo es el acceso a dispositivos de entrada/salida, como puede ser un disco duro. Varios procesos quieren usar ese componente, pero los accesos al mismo y su utilización están limitados, por lo que se acaba compartiendo el recurso y comienza a aparecer un tiempo de espera, un tiempo perdido que origina un cuello de botella. Una posible mejora de este rendimiento, intentando evitar la compartición de recursos, es la predicción de eventos, es decir, prever cuando un proceso va a solicitar acceso a un recurso e intentar gestionar esa petición con otro mecanismo.

Actualmente, son multitud las nuevas tecnologías que permiten prever eventos a partir de la minería de datos. Machine Learning, o aprendizaje automático, (Kelleher, Namee, & D'Arcy, 2015) es la más novedosa de ellas. A partir de un conjunto de datos, las máquinas son capaces de aprender y prever el comportamiento del sistema gracias al cálculo a partir de algoritmos sobre los datos recogidos de la probabilidad de que un evento ocurra. Fusionando estas tecnologías con herramientas sobre lectura y recogida de datos del rendimiento de los sistemas, se pueden desarrollar herramientas de mejora del rendimiento interesantes para estas plataformas.

En este documento se expone la idea de un trabajo fin de grado con el objetivo de crear una herramienta de aprendizaje automático que, a partir de una herramienta de lectura de rendimiento, pueda ampliar su funcionalidad para mejorar este rendimiento en caso de eventos de E/S. El proyecto tendrá los siguientes objetivos:

- Analizar y almacenar contadores hardware en busca de detectores de eventos de E/S.
- Analizar los algoritmos de Machine Learning en busca de un modelo de predicción de datos que se adecúe a nuestro problema y cumpla con los objetivos.
- Analizar las posibles soluciones, o mecanismos alternativos, para el caso de compartimiento de recurso.
- Analizar la mejora de rendimiento que ofrece la herramienta.
- Ofrecer posibles alternativas, o mejoras, a la herramienta para futuros trabajos.

Abstract

Performance is a very important factor in computing nowadays. Many market sectors, as well as investigators, are more focused on perfomance rather than in another system attribute. This is due to the increasing use of supercomputers and clusters, high performance computing systems with power based on parallel execution of great amount of nodes, process or threads. But this last generation systems are not infinite, they have limited resources and, on numerous occasions, they are not enough to reach the goal for what was used.

It is for this reason that seeks the best performance in this systems, ergo, to give the system a mechanism to improve, at the moment in which the resources are limited and have reached their limit, using to the maximum level the machine resources. For example, the access to a I/O element, such as a hard disk. Many process want to use this resource, but its access and use are limited, so finally the resource is shared, and it starts to appear a waiting time, a wasted time that could be used to execute another process. A posible performance improvement, avoiding shared resources, is the event prediction, i.e, anticipate the moment when a process is going to ask for an access to a busy resource, and trying to find another mechanism to do it.

Nowadays, there are many technologies that let you anticipate events based on data mining. Machine Learning is the newest technology for this. From a dataset, the machines are able to learn and anticipate the system behaviour thanks to the calculation, from this dataset and data algorithms, of a probability that an event will happen. Mixing this technologies with perfomance tools, improving perfomance tools can be developed, which are interesting for this macro systems.

In this document, the idea of a final grade work is exposed, with the goal of creating a machine learning tool that, from a perfomance tool, increase its funcionality to improve the performance in case of I/O events. The proyect will have these pre-objectives:

- Analyzing and save hardware counters looking for I/O events detectors.
- Analyzing Machine Learning algorithms seeking a data prediction model suitable to our problem and reaching all goals.
- Analyzing posible solutions, or alternative mechanisms, in case of resource sharing.
- Analyzing the performance improvement reached by the new tool.
- Offer possible alternatives, or improvements, for the tool in a future.

Contenido

1. Introducción 10
   1.1. Motivación 10
   1.2. Objetivos: principal y específicos. 11
   1.3. Estructura del documento 12
2. Estado de la cuestión 13
   2.1. Herramientas de monitorización de aplicaciones 13
   2.2. Contadores hardware 20
3. Desarrollo de la arquitectura. 22
   3.1. Entorno de desarrollo 22
4. Arquitectura propuesta 31
   4.1. Arquitectura original 31
   4.2. Arquitectura actual 34
   4.3. Análisis de requisitos 42
   4.4. Diagrama de clases e identificación de clases 50
   4.5. Definición de interfaces de usuario 50
5. Planificación 53
   5.1. Planificación temporal 53
   5.2. COCOMO 57
6. Evaluación 58
   6.1. Descripción de la plataforma 58
   6.2. Diseño de pruebas 59
   6.3. Análisis de los resultados 71
7. Presupuesto 79
   7.1. Mano de obra 79
   7.2. Hardware 80
   7.3. Software 80
   7.4. Material fungible 81
   7.5. Resumen 81
8. Conclusiones y trabajos futuros 82
   8.1. Conclusiones 82
   8.2. Trabajos futuros 84
9. Glosario de términos 85
10. Referencias 87

Anexo A: Manual de Usuario 89

Anexo B: Abstract 92

Índice de Ilustraciones

Ilustración 1. Interfaz de Vampir. 14
Ilustración 2. Interfaz gráfica de TAU. 15
Ilustración 3. Interfaz gráfica de JumpShot. 16
Ilustración 4. Interfaz gráfica de Paraver – pantalla de timeline. 18
Ilustración 5. Interfaz gráfica de Paraver – pantalla de estadísticas. 18
Ilustración 6. Interfaz de VTune Amplifier XE by Intel Corporation – visión global. 19
Ilustración 7. Interfaz de VTune Amplifier XE by Intel Corporation – análisis avanzado. 19
Ilustración 8. Diagrama de jerarquía de memoria caché. 20
Ilustración 9. Diagrama de arquitectura PAPI. 27
Ilustración 10. Diagrama de configuración del entorno de desarrollo. 30
Ilustración 11. Diagrama de arquitectura del sistema original. 31
Ilustración 12. Diagrama de arquitectura de la librería MPI original - inicialización. 32
Ilustración 13. Diagrama de arquitectura de la librería MPI original – lectura de contadores... 32
Ilustración 14. Diagrama de arquitectura del servidor original - inicialización. 33
Ilustración 15. Diagrama de arquitectura del servidor original – funcionamiento general. 33
Ilustración 16. Diagrama de arquitectura del sistema actual. 34
Ilustración 17. Diagrama de arquitectura de la librería actual - inicialización. 35
Ilustración 18. Diagrama de arquitectura del servidor actual – creación y ejecución del hilo. .. 36
Ilustración 19. Diagrama de la conexión original. 37
Ilustración 20. Diagrama de la conexión actual. 38
Ilustración 21. Diagrama del establecimiento de conexión del cliente actual. 38
Ilustración 22. Diagrama del establecimiento de conexión del servidor actual. 39
Ilustración 23. Diagrama de la arquitectura del servidor actual - inicialización. 39
Ilustración 24. Diagrama de la arquitectura del servidor actual – ejecución del hilo. 40
Ilustración 25. Interfaz de matriz de incrementos. 50
Ilustración 26. Interfaz de información global. 51
Ilustración 27. Pantalla de gráficas de contadores. 52
Ilustración 28. Diagrama de Gantt global. 53
Ilustración 29. Diagrama de Gantt - planificación. 54
Ilustración 30. Diagrama de Gantt – estudio del sistema base. 54
Ilustración 31. Diagrama de Gantt – estudio de viabilidad del sistema. 55
Ilustración 32. Diagrama de Gantt – análisis del sistema. 55
Ilustración 33. Diagrama de Gantt – diseño del sistema. 56
Ilustración 34. Diagrama de Gantt – implementación del sistema. 56
Ilustración 35. Diagrama de Gantt – pruebas. 57
Ilustración 36. Pseudocódigo gradient. 71
Ilustración 37. Gráfica de resultados. 77

Ilustración 38. Diagrama de la conexión actual – ingles. 95
Ilustración 39. Gráfica de resultados – inglés. 96

Índice de Tablas

Tabla 1. Plataformas de computación disponibles para Vampir. 13
Tabla 2. Patrón de requisito. 42
Tabla 3. Requisito RS-FU01. 43
Tabla 4. Requisito RS-FU02. 43
Tabla 5. Requisito RS-FU03. 43
Tabla 6. Requisito RS-FU04. 44
Tabla 7. Requisito RS-FU05. 44
Tabla 8. Requisito RS-FU06. 44
Tabla 9 Requisito RS-FU07. 45
Tabla 10. Requisito RS-FU08. 45
Tabla 11. Requisito RS-FU09. 45
Tabla 12. Requisito RS-FU10. 46
Tabla 13. Requisito RS-FU11. 46
Tabla 14. Requisito RS-FU12. 46
Tabla 15. Requisito RS-FU13. 47
Tabla 16. Requisito RS-FU14. 47
Tabla 17. Requisito RS-FU15. 47
Tabla 18. Requisito RS-FU16. 48
Tabla 19. Requisito RS-NF01. 48
Tabla 20. Requisito RS-NF03. 48
Tabla 21. Requisito RS-NF03. 49
Tabla 22. Requisito RS-NF04. 49
Tabla 23. Requisito RS-NF05. 49
Tabla 24. Patrón de prueba. 59
Tabla 25. Prueba PR-PF01. 60
Tabla 26. Prueba PR-PF02. 60
Tabla 27. Prueba PR-PF03. 60
Tabla 28. Prueba PR-PF04. 60
Tabla 29. Prueba PR-PF05. 60
Tabla 30. Prueba PR-PF06. 61
Tabla 31. Prueba PR-PF07. 61
Tabla 32. Prueba PR-PF08. 61
Tabla 33. Prueba PR-PF09. 61
Tabla 34. Prueba PR-PF10. 61

Tabla 35. Prueba PR-PF11. 62
Tabla 36. Prueba PR-PF12. 62
Tabla 37. Prueba PR-PF13. 62
Tabla 38. Prueba PR-PF14. 62
Tabla 39. Prueba PR-PF15. 62
Tabla 40. Prueba PR-PF16. 63
Tabla 41. Prueba PR-PF17. 63
Tabla 42. Prueba PR-PF18. 63
Tabla 43. Prueba PR-PF19. 63
Tabla 44. Prueba PR-PF20. 63
Tabla 45. Prueba PR-PF21. 64
Tabla 46. Prueba PR-PF22. 64
Tabla 47. Prueba PR-PF23. 64
Tabla 48. Prueba PR-PF24. 64
Tabla 49. Prueba PR-PF25. 64
Tabla 50. Prueba PR-PF26. 65
Tabla 51. Prueba PR-PF27. 65
Tabla 52. Prueba PR-PF28. 65
Tabla 53. Prueba PR-PF29. 65
Tabla 54. Prueba PR-PF30. 66
Tabla 55. Prueba PR-PF31. 66
Tabla 56. Prueba PR-PF32. 66
Tabla 57. Prueba PR-PF33. 66
Tabla 58. Prueba PR-PF34. 67
Tabla 59. Prueba PR-PF35. 67
Tabla 60. Prueba PR-PF36. 67
Tabla 61. Prueba PR-PF37. 67
Tabla 62. Prueba PR-PR01. 68
Tabla 63. Prueba PR-PR02. 68
Tabla 64. Prueba PR-PR03. 68
Tabla 65. Prueba PR-PR04. 68
Tabla 66. Prueba PR-PR05. 69
Tabla 67. Prueba PR-PR06. 69
Tabla 68. Prueba PR-PR07. 69
Tabla 69. Matriz de trazabilidad del sistema. 70
Tabla 70. Parámetros de configuración de pruebas de carga baja. 73
Tabla 71. Resultados de pruebas de carga baja – número reducido de procesos. 73
Tabla 72. Rendimiento de pruebas de carga baja – número reducido de procesos. 73

Tabla 73. Resultados de pruebas de carga baja – número medio de procesos. 73
Tabla 74. Rendimiento de pruebas de carga baja – número medio de procesos. 74
Tabla 75. Resultados de pruebas de carga baja – número alto de procesos. 74
Tabla 76. Rendimiento de pruebas de carga baja – número alto de procesos. 74
Tabla 77. Parámetros de configuración de pruebas de carga alta. 75
Tabla 78. Resultados de pruebas de carga alta – número bajo de procesos. 75
Tabla 79. Rendimiento de pruebas de carga alta – número bajo de procesos. 75
Tabla 80. Resultados de pruebas de carga alta – número medio de procesos 76
Tabla 81. Rendimiento de pruebas de carga alta – número medio de procesos. 76
Tabla 82. Resultados de pruebas de carga alta – número alto de procesos 76
Tabla 83. Rendimiento de pruebas de carga alta – número alto de procesos. 77
Tabla 84. Comparativa rendimiento entre tiempos de espera. 78
Tabla 85. Presupuesto – mano de obra. 79
Tabla 86. Presupuesto – hardware. 80
Tabla 87. Presupuesto – software. 80
Tabla 88. Presupuesto – material fungible. 81
Tabla 89. Presupuesto – resumen total. 81

1. Introducción

En este primer apartado realizaré una introducción a mi trabajo fin de grado. En él, se podrá observar la motivación de su realización, los objetivos, tanto principal como secundarios, del proyecto, y la estructura que va a seguir esta memoria.

1.1. Motivación

En la actualidad, el rendimiento en un factor muy importante en el ámbito empresarial y científico. El uso de supercomputadoras y clústeres es muy común en estos casos, por lo que la computación paralela es esencial para el funcionamiento de estos sistemas; ya que, al ejecutar programas con alto coste computacional y gran cantidad de instrucciones, es necesario el uso de varios nodos, procesos o hilos, para agilizar su ejecución.

El problema que reside en la mayoría de estos casos es el acceso a recursos dentro de una misma máquina. Cuando se realiza la computación paralela en una misma computadora, los procesos y los hilos tienen unos recursos finitos – y por tanto, pueden ocasionarse pérdidas de rendimiento o incluso de información.

La mayoría de estas pérdidas de rendimiento se encuentra a la hora de realizar instrucciones de entrada/salida en la ejecución de aplicaciones. Estas instrucciones se comunican, a través de buses, con la CPU. Es en el acceso a estas CPUs donde se encuentra el problema de rendimiento, ya que se puede encontrar el caso en el que dos o más hilos o procesos necesiten acceder a una misma CPU, y afectará negativamente al rendimiento y tiempo de ejecución de las aplicaciones.

El objetivo de este trabajo fin de grado es tratar este problema de pérdida de rendimiento a través de la predicción de estos eventos de entrada/salida mediante el uso de una técnica en auge actualmente, *Machine Learning*, es decir, usar inteligencia artificial para que aprenda los patrones de entrada/salida y, a partir de ese aprendizaje, poder deducir, o predecir, eventos futuros.

1.2. Objetivos: principal y específicos.

A continuación se detallarán los objetivos, tanto principales como específicos, que va a cumplir la herramienta para resolver el problema expuesto en el apartado anterior.

1.2.1. Objetivo principal

El objetivo principal es modificar la herramienta en la que está basado este trabajo, una herramienta de modelado y evaluación del rendimiento de aplicaciones; para que localice y prevea instrucciones de entrada y salida en las aplicaciones que evalúe; y tome medidas para evitar cuellos de botella en la compartición de recursos. Para ello, se pretende evaluar los eventos asociados a los contadores hardware que tiene el sistema, se observará qué contadores son los que indican que se está realizando una de estas instrucciones, y se realizará un modelado con *Machine Learning* para prever las siguientes.

1.2.2. Objetivos específicos

Los siguientes objetivos sirven de detalle y complemento al objetivo principal:

- Mejorar el sistema de comunicación cliente-servidor. El sistema actual de la aplicación es de sentido único cliente -> servidor. Nuestra aplicación necesita que se envía también información del servidor al cliente, para mandar medidas para evitar la compartición de recursos.

- Elección del algoritmo *Machine Learning* que mejor se adapte a nuestra aplicación. El aprendizaje automático posee infinidad de modelos de los que tenemos que encontrar el que mejor se adecue al problema y los datos que recoge el programa.

- Mostrar al usuario la información analizada en tiempo real. Para que el usuario interactúe con la herramienta, se le mostrará, como ya hacía la aplicación, los valores de los contadores hardware que recoge el programa.

- Detección de un patrón en los contadores que indique instrucciones de entrada y salida. Este patrón – ya sea aumento o bajada instantánea de los valores de algún contador – nos ayudará a observar si en esa aplicación se está realizando instrucciones de entrada y salida.

- Recogida de datos para el uso de *Machine Learning*. El programa recogerá los datos obtenidos en todos los ciclos de ejecución, para así poder proveer al modelo de *Machine Learning* de conocimiento que mejorará el modelo y la predicción.

- Reducción del tiempo de compartición de CPU entre aplicaciones, mediante la inducción de un tiempo de espera a las aplicaciones que vayan a usar una CPU ocupada.

1.3. Estructura del documento

En este apartado se especifica, en un pequeño resumen, la estructura del documento y el contenido de cada apartado que lo componen:

1. ***Introducción***: una breve presentación del trabajo, explicando los objetivos del mismo y las motivaciones que me han llevado a realizarlo.

2. ***Estado de la cuestión***: enumeración y descripción del software existente, y comparación de la herramienta con las distintas existentes.

3. ***Descripción de la arquitectura:*** se describirá el entorno de desarrollo utilizado (lenguajes de programación, sistema operativo, herramientas utilizadas), así como la arquitectura propuesta para la resolución del problema, junto con los diagramas de clases, requisitos e interfaces del sistema.

4. ***Planificación***: se realizará una estimación de la planificación del proyecto y el coste que esta tendrá.

5. ***Evaluación***: se presentarán las pruebas realizadas al sistema, para comprobar su buen funcionamiento, y comprobando que cumple con todo los requisitos estipulados.

6. ***Presupuesto***: descripción detallada del presupuesto estimado del proyecto.

7. ***Conclusiones y trabajos futuros***: resumen de los resultados obtenidos y esperados tras el desarrollo del proyecto, y una breve descripción de los posibles trabajos futuros que podrían realizarse a partir de este proyecto.

2. Estado de la cuestión

En este apartado se enumeraran algunas de las herramientas y aplicaciones que hay actualmente en el mercado de la monitorización de aplicaciones, y una pequeña introducción a los contadores hardware y su uso en el proyecto.

2.1. Herramientas de monitorización de aplicaciones

Como se ha especificado en la introducción, el rendimiento es un factor muy importante a la hora de ejecutar macro programas en varios hilos o procesos paralelos, y gracias a estas herramientas podemos monitorizar ese rendimiento. El objetivo principal de estas herramientas es buscar y mostrar dónde se encuentran las pérdidas de rendimiento – ya sea por acceso a un mismo recurso, fallo de comunicación, algún fragmento de código ineficiente o con algún tipo de fallo – y buscar, así, un modo de que el tiempo de ejecución sea mínimo y, por tanto, tengamos un programa lo más eficiente posible.

Aquí se mostrarán algunas de las herramientas que más se utilizan para este fin, y un pequeño resumen de su funcionalidad.

2.1.1. Vampir

Actualmente en la versión 9.1, la herramienta *Vampir*, diseñada en el Centro de Matemáticas Aplicadas del Centro de Investigación Jülich y el Centro de Computación de Alto Rendimiento de la Universidad Técnica de Dresde, ha sido uno de los mayores exponentes en herramientas de monitorización comerciales del mundo; presente en muchísimas investigaciones y proyectos de desarrollo. (GWT-TUD GmbH, 2016)

Una de las ventajas de usar *Vampir* es la capacidad de leer ficheros de rastreo de otras herramientas como son TAU, KOJAK, *VampirTrace* o OTF – especialmente diseñado para programas con computación paralela masiva – pero, desde 2005, no puede leer los ficheros de rastreo de Intel, ya que en ese año terminó su cooperación y por motivos de licencia no lo tienen permitido. También permite la interceptación y grabación de eventos de comunicación MPI de programas paralelos.

Otra ventaja que tiene *Vampir* es la gran portabilidad que posee gracias a su gráfica basada en interfaz de usuario, haciendo que esté disponible en muchas plataformas de computación:

Plataforma	**Versiones**		
Linux/UNIX	IA32	IA64	IA86_64
Windows	X32	X64	
Mac OS X	Universal		

Tabla 1. Plataformas de computación disponibles para Vampir.

Su *Framework* posee las siguientes características y funcionalidades:

- Permite hacer un detallado zoom y desplazamiento en todas las pantallas.
- Ofrece estadísticas con posibilidad de adaptarse al usuario en rangos de tiempo seleccionados.
- Es fácil de usar a la hora de hacer análisis de rendimiento en programas paralelos.
- La representación gráfica de los datos permite entender detalladamente los procesos dinámicos en sistemas paralelos masivos.
- Análisis en profundidad del comportamiento en tiempo de ejecución de los procesos en paralelo y de la comunicación entre procesos.
- Filtrado de procesos, funciones, mensajes…
- Agrupación jerárquica de hilos, procesos y nodos.
- Identificación de problemas de rendimiento y cuello de botella.
- Captura de pantalla y copias para publicaciones integradas.
- Pantallas personalizables.

Además, incluye una sub-herramienta, *VampirServer*, la cual permite:

- Rediseño ultra escalable de la funcionalidad establecida por *Vampir*.
- Visualización de manera distribuida de los datos sobre el rendimiento.
- Incrementada escalabilidad comparado con el enfoque secuencial.
- Navegador de datos de rendimiento desde sitios remotos.

A continuación resumiré brevemente su interfaz, la cual es muy intuitiva y gráfica, la cual la hace simple a la vez que completa:

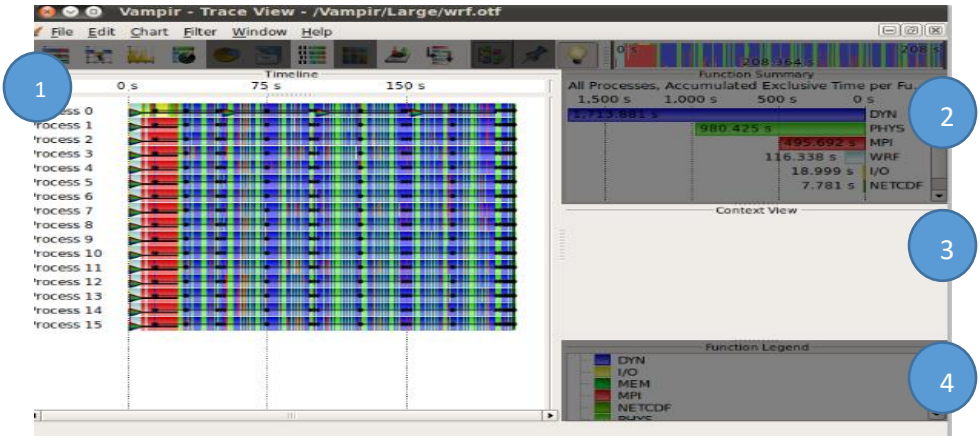

Ilustración 1. Interfaz de Vampir.

1) Gráfica que representa el análisis de rendimiento basado en eventos, es decir, la cadena de eventos del proceso, o contadores en un eje de tiempo horizontal.
2) Resumen de las funciones, que da una visión del consumo de tiempo acumulado en todos los grupos de funciones.
3) La pestaña de contexto proporciona información más detallada de un objeto seleccionado en comparación con su representación gráfica.
4) Leyenda de las funciones, que enumera los grupos de funciones junto al color que les representa en la gráfica.

2.1.2. TAU

TAU Performance System es un conjunto de herramientas portable para el seguimiento y análisis del rendimiento de programas paralelos escritos en Fortran, C, C++, UPC, Java y Python. Desarrollada por *DOE Office of Science*, iniciativa ASC en LLNL, el proyecto *ZeptoOS* en ANL y *el National Laboratory* en Los Álamos, TAU provee una gran variedad de herramientas dinámicas y estáticas para dar interacción de usuario al análisis del ambiente de las aplicaciones paralelas. (Shende & Malony, Summer 2006)

Esta aplicación puede reunir información sobre el rendimiento mediante el análisis de las funciones, métodos, bloques y los estados, además de muestrear los eventos. Una ventaja de TAU es que soporta todas las características de C++, incluyendo las plantillas y los *namespaces*.

Los instrumentos que contiene TAU pueden ser insertados en el código fuente mediante el uso de una herramienta de instrumentación automática basada en *Program Database Toolkit* (PDT), que usa dinámicamente *DyninstAPI*, en tiempo de ejecución en la Máquina Virtual de Java, o manualmente usando la instrumentación API.

La herramienta de visualización de TAU, *paraprof*, muestra una pantalla gráfica de todos los resultados de los análisis de rendimiento, en conjunto o en nodo, contexto o hilo en particular. Así, el usuario puede identificar rápidamente las fuentes de los cuellos de botella en la aplicación gracias a esta interfaz gráfica. Además. Genera trazas de eventos que pueden ser mostradas con otros programas como *Vampir, Paraver* o *JumpShot.*

A continuación podremos observar la interfaz gráfica de TAU, dividida en nodos e hilos de cada nodo, y los valores de los temporizadores de cada hilo.

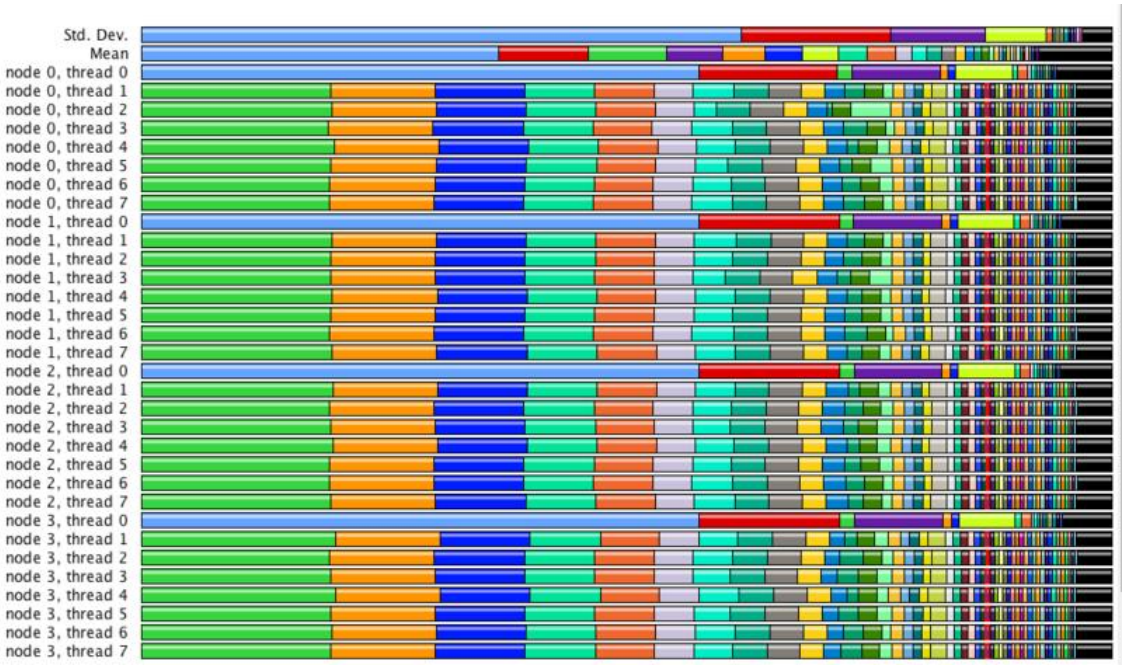

Ilustración 2. Interfaz gráfica de TAU.

2.1.3. JumpShot

JumpShot es una herramienta de visualización basada en Java para el análisis post-mortem de rendimiento. El beneficio de ser una herramienta basada en Java es, mayoritariamente, su portabilidad, mantenimiento y funcionalidad de la herramienta. Actualmente se encuentra en su versión 4, la cual explicaremos a continuación. (LANS, 2007) La finalidad de esta herramienta es poder observar los archivos de trazas que han creado otras herramientas, como *Vampir* o TAU. Estos ficheros contienen secuencias de eventos, con un tiempo e información en ellos; y una colección de estos eventos es un *timeline*, o línea temporal. El análisis port-mortem que realiza *JumpShot* está basado en estos ficheros, siendo una gran ayuda para la visualización del rendimiento de programas paralelos.

JumpShot4 permite un gran nivel de abstracción de los detalles sin tener que leer una gran cantidad de información como la que se suelen mostrar en las gráficas. También permite hacer zoom a zonas específicas, o al conjunto; y moverse por las líneas temporales de cada hilo. También provee un control central tanto del histograma como del *timeline* global, y se puede realizar búsquedas o escáneres para localizar objetos difíciles de encontrar en los *logfiles* de gran tamaño.

Otra funcionalidad que tiene *JumpShot* es la visualización única de eventos MPI, eliminando de los *timelines* los eventos no MPI. Y, por último, nos ofrece un conversor de formatos de trazas, haciendo de esta herramienta una de las más escalable.

En esta imagen podemos observar tanto la funcionalidad del zoom en una zona concreta, como la gráfica *timeline* que nos ofrece la herramienta, mostrándonos la ejecución de cada proceso en un periodo de tiempo:

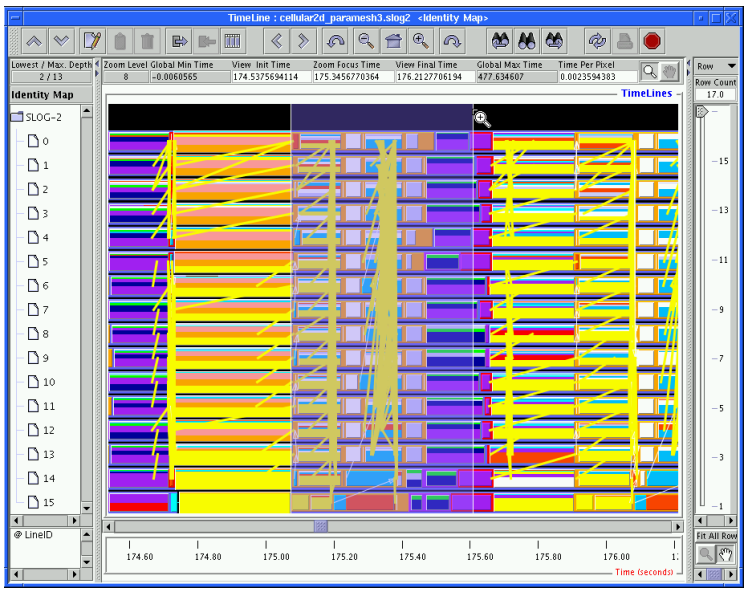

Ilustración 3. Interfaz gráfica de JumpShot.

2.1.4. Paraver

Desarrollado por el BSC (Barcelona Supercomputing Center, 2016), Paraver tiene como objetivo de responder a la necesidad de tener una percepción global cualitativa del comportamiento de la aplicación mediante la inspección visual, y le enfoque detallado del análisis cuantitativo de los problemas, tanto de rendimiento como errores de código. Todo ello dio resultado en una herramienta de análisis del rendimiento basado en trazas, con una gran flexibilidad para explorar los datos recogidos. Está contenido dentro del kit de herramientas CEPBA-Tools.

Paraver es un buscador de información muy flexible. Su análisis se centra en dos pilares: el primer es que su formato de traza no tiene semántica, lo cual es una ventaja a largo plazo, ya que a la hora de adaptarse a los nuevos modelos de programación o nuevos datos de rendimiento no necesita adaptar ni modificar el visualizador.

El segundo pilar es que las herramientas no están conectadas a la herramienta, si no que están programadas en ella. Para ejecutarlas, la herramienta posee una gran variedad de funciones de tiempo, módulos de filtrado y mecanismos para combinar *timelines*. Este enfoque de la herramienta permite mostrar un gran número de métricas con la información disponible. Esto permite que se guarden las vistas, o conjuntos de vistas y, a la hora de volver a computar la vista con nueva información solo sería necesario cargar el archivo guardado.

Algunas características de *Paraver* son:

- Presentación de muchos más detalles que otras herramientas de rendimiento.
- Análisis cuantitativo detallado del rendimiento del programa.
- Análisis comparativo recurrente de múltiples trazas.
- Semántica personalizada de la información visualizada.
- Trabajo cooperativo, compartiendo vistas del fichero de trazas.
- Construcción de métricas derivadas.

Paraver también ofrece un conjunto mínimo de vistas en una traza. Estas vistas presentan la información de rendimiento en dos pantallas principales que muestran diferente tipo de información: la pantalla de *timeline* representa el comportamiento de la aplicación proceso/tiempo, y la pantalla estadística representa el análisis numérico de la información de una región seleccionada, ayudando a sacar conclusiones sobre la optimización.

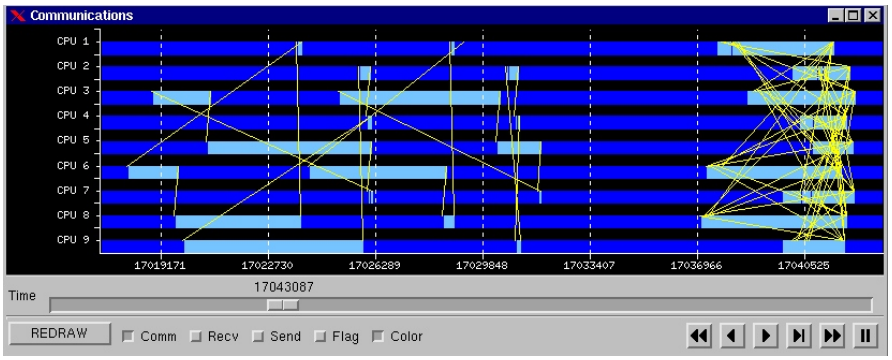

Ilustración 4. Interfaz gráfica de Paraver – pantalla de timeline.

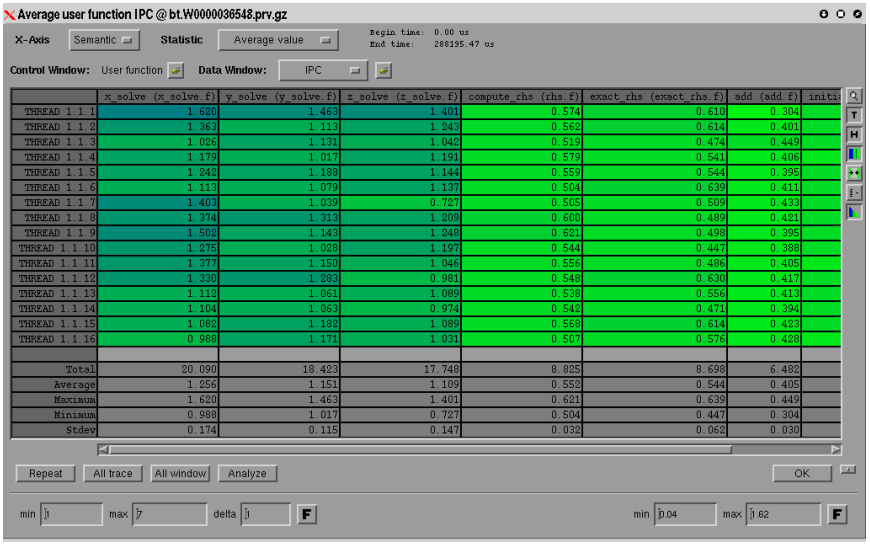

Ilustración 5. Interfaz gráfica de Paraver – pantalla de estadísticas.

La información mostrada en *Paraver* se resume en tres elementos: un valor semántico, dependiente del tiempo, para cada objeto representado; *flags* que corresponden a eventos puntuales, y líneas que representa la comunicación de los objetos representados. El módulo de visualización determina cómo se muestran estos elementos, así como cambiar su representación, colores y escalas; además de representar los valores y los eventos, sin asignarles ninguna semántica preconcebida; dándole flexibilidad a la herramienta.

La última ventaja de *Paraver* es que permite comparar múltiples trazas, como dos versiones del mismo código, el comportamiento en dos máquinas, las diferencias entre ejecuciones, etc.

2.1.5. VTune Amplifier XE by Intel Corporation

Esta herramienta, desarrollada por Intel, es un creador de perfiles de rendimiento para diversos lenguajes como son C, C++, C#, Fortran, Assembly y Java; analizando tanto aplicaciones en MPI como OpenMP. (Intel, 2016)

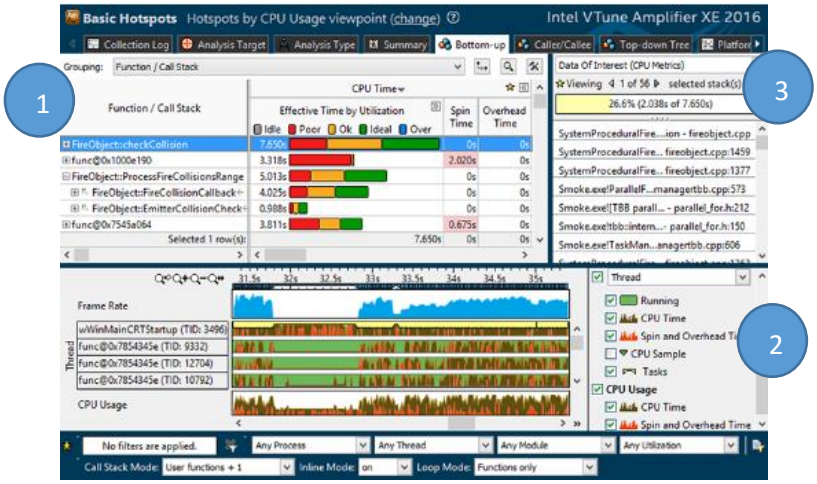

Ilustración 6. Interfaz de VTune Amplifier XE by Intel Corporation – visión global.

1) El análisis de puntos críticos ofrece una lista de funciones que usa el mayor tiempo de CPU. Si se realiza doble clic sobre la función de la lista te lleva al punto crítico dentro de esa función.
2) Rango temporal que permite analizar los datos usando un filtro de uso de CPU respecto al tiempo. También permite realizar un análisis de paradas y esperas. Esto permite encontrar la causa de un fallo en el rendimiento en programas paralelo, debido por una espera extensa en una parada, lo cual quiere decir que durante ese tiempo los núcleos quedan inutilizados.
3) También nos ofrece algún dato de interés, como métricas de CPU o el porcentaje de tiempo que usa un grupo de funciones.

Finalmente, *VTune* nos ofrece también análisis más avanzados, que cuenta directamente con varias opciones: uso de ancho de banda, acceso a memoria, predicción de fallos, etc.; además de permitir crear perfiles establecidos para un inicio rápido, o elegir un perfil para zonas críticas, concurrencia, etc.

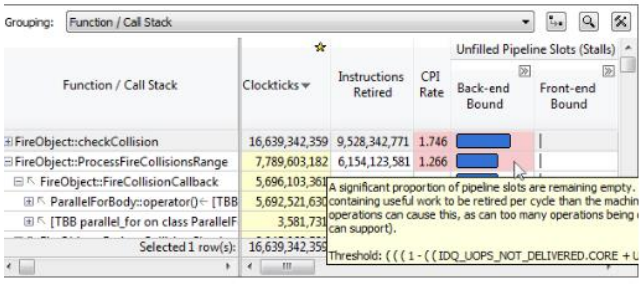

Ilustración 7. Interfaz de VTune Amplifier XE by Intel Corporation – análisis avanzado.

2.2. Contadores hardware

Los contadores hardware van a ser el centro de este proyecto, por lo que hay que saber con qué se está tratando. Todo procesador tiene registros, es decir, localizaciones rápidamente accesibles a las CPU, los cuales son de solo escritura o solo lectura. Hay registros de datos, de direcciones, de estado… pero en los procesadores actuales hay presentes un grupo de registros para información especial.

Este grupo de registros son los contadores hardware, a los que se pueden acceder sin coste adicional para la CPU; y sirven para medir el rendimiento y otra información interesante para la optimización y elaboración de estadísticas acerca del programa que se está ejecutando, y la máquina en la que se está ejecutando.

Estos registros, y su nombrado, difieren de cada modelo de procesador, lo cual hace muy problemática su familiarización y su uso. Por ello, se establecieron unos eventos básicos comunes a todos los procesadores, para facilitar la programación de estos registros, ya que son los eventos más relevantes y útiles a la hora de hacer evaluación de rendimiento.

Estos eventos básicos dan acceso a:

- ***Caché***:

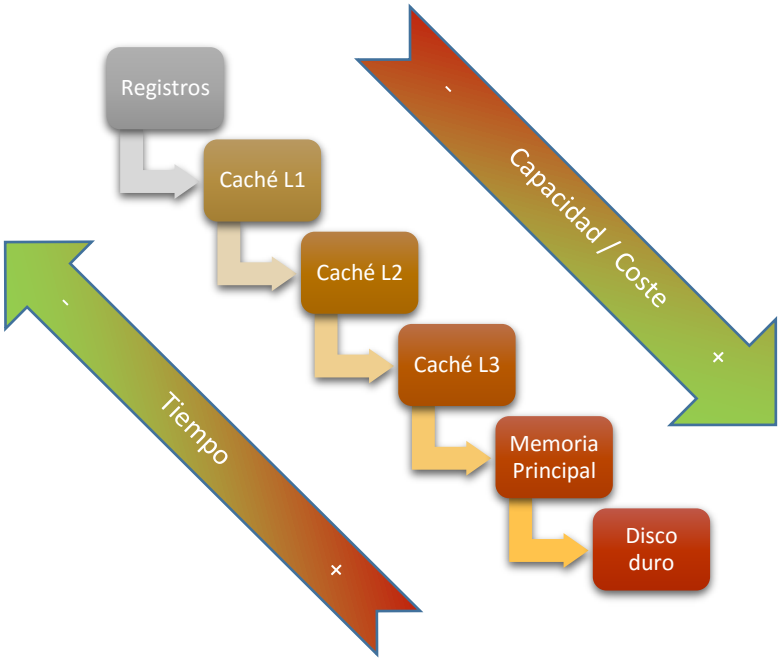

Ilustración 8. Diagrama de jerarquía de memoria caché.

La memoria caché sirve para acceder a datos útiles, o usados anteriormente, a mayor velocidad que accediendo a ellos directamente desde memoria principal. Esto permite una ejecución más rápida debido a la mayor optimización de rendimiento del programa. Este tipo de eventos se encargan de contabilizar y recoger estos accesos y los eventos que tienen detrás. Estos accesos son jerárquicos, es decir, la CPU necesita datos para operar, y si no los tiene, primero accede a la caché L1, para comprobar si posee ese dato y copiarlo en sus registros.

Si no se da ese caso, se accede a un nivel más debajo en la jerarquía, la caché L2, y así hasta encontrar el dato por todos los niveles que tenga la máquina. Si no se encuentra en ninguno de estos niveles, se accede ya a memoria principal. A cada nivel más bajo en la jerarquía, mayor penalización de coste computacional hay; y por tanto, menor rendimiento.
Por lo tanto, estos registros guardan estos accesos a los distintos niveles de la jerarquía de memoria, es decir, el número de aciertos en cada nivel (veces que el dato estaba en el nivel), el número de fallos en cada nivel (veces que no estaba el dato en el nivel), etc.

- ***Ciclos***: los registros contabilizan los ciclos que ocupan o tarda una actividad en realizarse, por ejemplo, los ciclos que una aplicación usa para realizar operaciones en coma flotante, esperando acceso a memoria, etc.

- ***Instrucciones***: los registros contabilizan las instrucciones que se ejecutan durante el programa y los eventos que esto conlleva. Por ejemplo, al no tener todas las instrucciones el mismo coste computacional, hay registros para los FLOPS (operaciones en punto flotante), de precisión simple, de saltos – en este caso se registrarían los saltos tomados, predichos y no tomados -, y muchos más tipos de operaciones.

Además de estos registros básicos, hay muchos más dependiendo de la arquitectura, niveles de caché que tenga la computadora, registros de entrada/salida, o incluso de bus de memoria. También, gracias a estos eventos básicos, se pueden obtener eventos derivados; por ejemplo, si tenemos los eventos simples de calcular el número de aciertos y fallos de la caché, podremos calcular el número de accesos total a la caché.

Algunas librerías, como **PAPI** (*Performance Application Programming Interface*), poseen aproximadamente 100 eventos preestablecidos. Esto permite una monitorización eficiente y sencilla de los múltiples contadores que posee cada procesador, siendo la librería más portable del mercado, ya que lo permite en múltiples sistemas operativos y diversas plataformas. (ICL UR, 2016).

También está el caso de la herramienta **perf**, incluida en el núcleo de Linux y en continua actualización, la cual te permite acceder a contadores de rendimiento, *tracepoints* – puntos establecidos a lo largo del código, como en llamadas al sistema, para recopilar información en un tiempo o traza establecidos - , *kprobes y uprobes* – *tracepoints* dinámicos, para trazado de *kernel* (núcleo) o *userspace* (espacio de usuario). (Wikimedia foundation, 2016)

3. Desarrollo de la arquitectura.

La aplicación a desarrollar tiene como objetivo ampliar el funcionamiento actual de la herramienta de monitorización desarrollada por Roberto García; para permitir una monitorización centralizada que detecte cuando una o varias aplicaciones paralelas están realizando instrucciones o eventos de entrada/salida; y mediante un algoritmo de *Machine Learning* acabe prediciendo este tipo de eventos y evitando así retardos a la hora de acceder al recurso, en este caso HD.

3.1. Entorno de desarrollo

En este apartado se especificará el entorno de desarrollo. Como es un proyecto basado en uno anterior, únicamente se describirán los entornos usados en este trabajo, a excepción de las librerías externas, en las que describiré tanto las usadas como el motivo por el que se han utilizado la librería de *Machine Learning*

3.1.1. Lenguaje de programación

En un proyecto el lenguaje de programación se establece con el objetivo de garantizar tanto el cumplimiento de la necesidad del proyecto y del sistema, además de poder garantizar una portabilidad de esta aplicación para su uso en varios sistemas. Estos motivos han sido clave para la elección de los siguientes lenguajes de programación:

C

El lenguaje de programación C, desarrollado por Dennis M. Ritchie, es un lenguaje de estructuras de alto nivel que a la vez permite la manipulación a bajo nivel, orientado a la implementación de sistemas operativos. Es, además uno de los lenguajes más eficientes y portables del mercado, ya que tiene compiladores para casi todos los sistemas operativos. (Wikimedia foundation, 2016)

El lenguaje C tiene también las siguientes características:

- Tiene un núcleo de lenguaje simple, pero este se puede ampliar y añadir funciones más complejas e importantes gracias al uso de bibliotecas.
- Como he mencionado antes, tiene gran portabilidad gracias a los múltiples compiladores que existen para diversos sistemas operativos.
- También es un lenguaje de bajo nivel, lo cual permite realizar implementaciones más óptimas, aumentando su eficiencia.
- Existen compiladores en los principales sistemas operativos descargables de forma gratuita.

Java

Java es uno de los lenguajes de programación más populares en la actualidad. Diseñado por *Sun Microsystems*, es un lenguaje de programación orientado a objetos, y concurrente, diseñado para tener tan pocas dependencias de implementación como fuera posible. (Oracle, 2016)

Es uno de los lenguajes de programación más populares, también, para la programación de aplicaciones cliente-servidor. Esto es debido a la gran portabilidad que posee Java, ya que está diseñado para que se escriba el programa una única vez y se ejecute en cualquier dispositivo, sin la necesidad de ser recompilado; y también por la gran cantidad de usuarios que lo utilizan a lo largo del mundo.

Las características principales de java son:

- Lenguaje totalmente orientado a Objetos, es decir, todos los tipos, excepto las variables fundamentales, son clases.
- Amplio catálogo de bibliotecas, al igual que C, que permite añadir funciones más complejas, e incluso clases completas, que permiten realizar cualquier tipo de aplicación.
- Java es distribuido, gracias a las clases que proporciona para su uso en red, así como la creación de sockets y permitir la comunicación cliente-servidor.
- Es un lenguaje robusto y seguro, ya que fue diseñado para crear software altamente fiable, y comunicaciones entre cliente-servidor seguras, mediante comprobaciones en compilación, tiempo de ejecución y con barreras de seguridad en el tiempo de ejecución.
- Es independiente de la arquitectura en la que se ejecute, ya que está diseñado para que se ejecute tanto en Windows como en Linux o Mac, gracias a la generación de *bytecodes*, el cual lo compila y ejecuta el intérprete de java.
- Permite el *multithreading*, con la ejecución de múltiples hilos y la comunicación y sincronización entre ellos, lo cual permite la distribución mencionada anteriormente.
- Es portable.
- Es un lenguaje simple, muy parecido a C.

3.1.2. Sistema operativo

En este apartado se enumeraran los Sistemas Operativos empleados en el desarrollo del proyecto, además de una descripción de su funcionalidad y sus características. A la hora de elegir estos sistemas operativos se debe tener en cuenta su popularidad, su disponibilidad y el conocimiento sobre el sistema.

Los sistemas operativos utilizados en el proyecto han sido los siguientes:

GNU / Linux

Sistema operativo basado en el código libre y gratuito, a partir de la idea de Richard Stallman y desarrollado por Linus Torvalds; y dispone de varias distribuciones por la red (Ubuntu, Debian, etc.). Es un sistema operativo presente en muchos hogares e instituciones, así como en los computadores de la universidad. (The Linux foundation, 2016)

Las ventajas que nos ofrece este sistema para nuestro proyecto son:

- Sistema operativo gratuito y abierto.
- Disponibilidad en la Universidad, tanto del sistema operativo como de las librerías a utilizar.
- Sistema fiable y ofrece facilidades respecto a la programación en C.
- Ofrece una consola en la que se puede ejecutar y manejar los programas.

Microsoft Windows

Desarrollado por Microsoft, es el sistema operativo más popular y usado del mercado, disponible en la amplia mayoría de dispositivos. Al contrario que Linux, esta distribución necesita una licencia de pago. (Microsoft, 2016)

Las ventajas que nos ofrece Windows son:

- Está disponible en todos los equipos, tanto de la universidad como en los personales.
- Posee programas para el desarrollo de aplicaciones Java, y la ejecución de estos.
- Su interfaz es práctica, intuitiva y sencilla.

3.1.3. Herramientas de desarrollo

En este apartado se describirán los softwares utilizados para la programación y desarrollo de la aplicación en los sistemas operativos especificados. Para ello, seguiremos la base del software usado durante la carrera, ya que la experiencia en ellos será superior a otras herramientas. Por ello, para la programación en Java se utilizará **Eclipse** (The Eclipse Foundation, 2016), y para la programación en C, dependerá del sistema operativo en el que se programe: en el caso de Windows, se usará el programa **Sublime Text 3** (Sublime, 2016); y en el caso de Linux, **Gedit** (The GNOME Project, 2016)

Eclipse

Plataforma de software multiplataforma, creado por IBM y desarrollado por la Fundación Eclipse, compuesto por herramientas para programar aplicaciones y entornos de desarrollo integrado. Dispone de un editor de texto, con compilación en tiempo real, posibilidad de realizar pruebas unitarias y añadir más funcionalidades a través de *"plugins"*. También cabe la posibilidad de ampliar los lenguajes de programación a C/C++ y Python. Por último, dispone de una interfaz gráfica muy intuitiva que permite al usuario una experiencia más sencilla.

Sublime Text 3

Editor de texto y de código fuente, escrito en C++ y Python, de descarga gratuita pero con licencia de pago para uso continuado. Tiene como características destacables un mini mapa para previsualizar la estructura del código, da soporte para 43 lenguajes, entre los que están Java, C, C++, etc.; y permite, gracias a muchas funcionalidades, la edición y programación intuitiva.

Es únicamente un editor de código fuente. A diferencia de Eclipse, para ejecutar los programas que se están desarrollando es necesario usar el compilador de C integrado en Linux.

Gedit

Editor de texto para Linux, que incluye herramientas para la edición de código fuente y textos estructurados. Es el predeterminado para la distribución GNOME, y está distribuido bajo las condiciones de software libre. Se destaca su facilidad de uso y su coloreado e interfaz con pestañas para facilitar la múltiple edición. Además, incluye un corrector multilenguaje y un sistema de *plugins* que amplía la funcionalidad de la aplicación.

3.1.4. Librerías externas

En este apartado se comentarán y describirán las librerías utilizadas en la realización de este proyecto. Gracias a estas librerías es posible realizar la funcionalidad del proyecto especificada, así como cumplir los objetivos detallados anteriormente.

PAPI

El proyecto PAPI (*Performance Application Programming Interface*), originado en el Laboratorio de Computación Innovadora de la Universidad de Tennessee, específica una interfaz estándar de programación de aplicaciones (API), útil para acceder a los contadores de rendimiento que están disponibles en la mayoría de los microprocesadores actuales. Estos contadores existen como un pequeño conjunto de registros que cuantifica eventos, señales especificas relacionadas con la función del procesador. La monitorización de estos eventos facilita la correlación entre el código y la estructura que lo ejecuta, siendo útil en muchísimos casos de análisis de rendimiento, incluyendo *debugging*, monitorización, modelado de rendimiento, etc. (ICL UR, 2016)

PAPI posee dos interfaces para los contadores hardware: una simple, de alto nivel, para la adquisición de simples medidas y totalmente programable; y una interfaz de bajo nivel destinada a usuarios con necesidades más sofisticadas. Esta última interfaz es la que lidia con los eventos hardware en grupos llamados *EventSets*, los cuales reflejan el uso mayoritario de estos contadores, así como tomar medidas simultáneas de diferentes eventos hardware y relacionarlas. Estos *EventSets* son totalmente programables, y poseen características como:

- Protección y seguridad de los hilos garantizada.
- Escritura de valores de contadores.
- Multiplicación y notificación en el cruce de umbral.
- Además, poseen características propias de cada procesador.

La interfaz de alto nivel tiene la capacidad de modificar el estado de eventos específicos, así como leer su contenido; pero solamente uno a cada momento.

Otra característica de PAPI es la gran portabilidad en las diferentes plataformas. Usa las mismas rutinas con una lista de argumentos similar para controlar y acceder a los contadores de todas las arquitecturas. Como parte de PAPI, se han predefinido un grupo de eventos que representa los eventos más comunes de toda buena implementación, con el objetivo de conseguir un conteo similar de los mismos eventos en distintas plataformas. Esto depende del programador, si decide usar este conjunto de eventos no necesitará modificar el código cuando lo vaya a ejecutar en otra plataforma; pero, si el programador decide usar eventos específicos, la API de bajo nivel da acceso a todos los eventos disponibles y, si este evento no lo estuviera en la máquina en la que se está ejecutando, devolverá un código de error. Esto reduce el esfuerzo de migrar el código, debido a que no tiene que actualizar cada llamada a PAPI, ya que sigue siendo la misma en todos los casos; lo único que actualiza es la lista de argumentos.

Además de este set de eventos estándar, cada implementación de PAPI soporta todos los eventos nativos a través de la habilidad de aceptar contadores específicos de cada plataforma. Estos contadores vienen especificados en el fichero de cabecera. Así se evita tener código ineficiente para trasladar todos los eventos de todas las plataformas en el código.

PAPI también puede ser dividido en dos capas de software. La capa superior (*Framework Layer*) consiste en la API y la máquina independientemente de las funciones que soporta. La capa inferior (*Component Layer)* define y exporta una interfaz de máquina independiente a las funciones y estructuras de datos dependientes de la máquina. Estas funciones acceden al sistema operativo, una extensión del *kernel* o funciones que acceden directamente a los registros del procesador. PAPI intenta usar el mejor de esos tres casos siempre, con el fin de buscar el caso más eficiente y flexible, dependiendo de su disponibilidad. Además, la funcionalidad de las capas superiores dependen en gran medida de lo que se escoja en este nivel y, si este nivel no da una funcionalidad disponible, se trata de emular esta funcionalidad. A continuación se muestra, de forma gráfica, la arquitectura de PAPI descrita anteriormente:

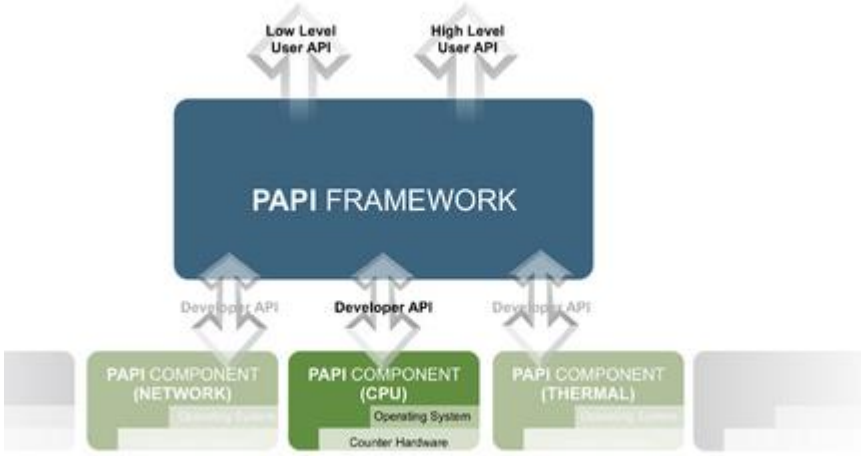

Ilustración 9. Diagrama de arquitectura PAPI.

PAPI intenta proveer una funcionalidad uniforme entre plataformas. Aun así, esto no es siempre posible, por lo que, si el hardware no soporta algunas características, PAPI intenta que estas sean soportadas por el software. Lo mismo pasa con las métricas, no todas pueden ser soportadas por el procesador, dependiendo estas del modelo que se esté usando.

PAPI también se asegura que el sistema operativo o la misma librería estén protegidos ante el desbordamiento del de los valores de los contadores. Cada contador puede ser incrementado más de una vez en un único ciclo de reloj. Esto, combinado con el aumento de las velocidades de este reloj y la poca precisión de algunos contadores físicos significa que es muy sencillo que este desbordamiento ocurra. Esto se evita con la capacidad del usuario de crear controladores para manejarlos desde el *hardware* (mediante interrupciones) o *software* (temporizador/controlador).

Por último, PAPI provee de una implementación portable de notificaciones asíncronas cuando los contadores sobrepasan un valor especificado por el usuario. Esta funcionalidad provee las bases para llamadas compatibles *SVR4*, que generan un histograma detallado de las interrupciones en el rendimiento basado en métricas hardware, no en tiempo.

Para resumir, PAPI es la librería más completa y portable que provee acceso a los contadores hardware, además de proveer de otras características a la hora de usarse en diversas máquinas.

MPI

MPI (*Message Passing Interface* – Interfaz de Paso de Mensajes) es un estándar con el objetivo de definir la sintaxis y la semántica de las funciones accesibles a través de una biblioteca de paso de mensajes, con el fin de crear programas distribuidos en múltiples procesadores. (MPICH Project, 2016)

MPI no es una implementación, es una interfaz de comunicaciones y sincronizaciones, lo cual lo hace independiente al lenguaje de programación en el que esté escrito el programa paralelo. Esta interfaz ha de seguirse en el comportamiento de toda implementación en sistemas paralelos, soportando tanto comunicaciones punto a punto, como colectivas; pero detalla solo las funciones que se han de utilizar, no el modo en el que se compilen y ejecuten (lo cual puede variar entre implementaciones). Estas funciones son:

- Iniciar, gestionar y finalizar procesos MPI.
- Comunicación entre dos procesos.
- Operaciones de comunicación entre grupos de procesos.
- Crear tipos de datos.
- Sincronización.

Estas funciones se llaman directamente desde C, C++, Fortran o cualquier lenguaje con interfaz en dicha librería (C#, Java, Python, etc.)

Además, MPI tiene como características:

- La estandarización del paso de mensajes.
- La portabilidad: soporta multiprocesadores, multicomputadores, redes, sistemas heterogéneos, etc.
- Ofrece buenas prestaciones y una amplia funcionalidad.
- Existen implementaciones libres, como mpich.

JFreeChart

El proyecto (Gilbert, 2016), es la librería gráfica más usada actualmente para Java, con más de 2.2 millones de descargas hasta la fecha (2014). Es una librería gráfica 100% gratuita, lo cual permite a los desarrolladores mostrar gráficas de calidad profesional en sus aplicaciones.

Las características principales de JFreeChart son:

- Es una API consistente y bien documentada, dando soporte a un gran rango de gráficas.
- Posee un diseño muy flexible, lo que la hace fácil de extender y añadir nuevas funcionalidades. Además, se enfoca tanto en aplicaciones de servidor como de cliente.
- Da soporte a muchos ficheros de salida, incluyendo imágenes, formatos de vectores gráficos, etc.
- Es gratuito.

Libxml2

Libxml2 es un kit de herramientas de desarrollo y *parser* de XML para C, desarrollado por el proyecto GNOME, siendo un software libre disponible bajo la licencia MIT. (The GNOME Project, 2016)

XML en si es un metalenguaje para diseñar lenguaje con etiquetas, por ejemplo, un lenguaje de texto donde la semántica y la estructura son añadidas al contenido usando información extra por etiquetas, escritas entre “< >”. El lenguaje similar más conocido es HTML.

Libxml2 está escrito en C, pero está disponible en distintos lenguajes. Esto la hace muy portable, por lo que la librería debería poder ser instalada y ejecutada sin problemas en distintos sistemas (Linux, Windows, MacOS, etc.).

Weka

El proyecto WEKA (*Waikato Environment for Knowledge Analysis* – Entorno para análisis de conocimiento de la Universidad de Waikato) es una plataforma software para el aprendizaje automático y la minería de datos. (Hall, y otros, 2009)

Se compone de dos elementos: una aplicación gráfica desde la que se pueden realizar experimentos y minería de datos; y lo que nos interesa a nosotros, una librería que contiene una colección de herramientas de visualización y algoritmos para análisis de datos y **modelado predictivo**.

Tiene como características:

- Está disponible libremente bajo la licencia pública general de GNU.
- Es muy portable, ya que esta implementado en Java y puede ejecutarse en casi cualquier plataforma.
- Es muy simple: todas las funcionalidades que encontramos en su interfaz gráfica se pueden acceder mediante un simple par de llamadas a funciones de la biblioteca.
- Soporta tareas estándar de minería de datos: preprocesamiento de datos*, clustering*, clasificación, regresión, visualización, selección, predicción…
- Proporciona acceso a bases de datos como son SQL, arff, etc.

Se ha escogido esta herramienta para las tareas de *Machine Learning*, ya que se puede experimentar, previo a la implementación, con los datos recogidos de la aplicación y poder observar que modelo de datos y predicción es el adecuado para alcanzar nuestros objetivos.

3.1.5. Configuración del entorno de desarrollo

Tras lo especificado anteriormente, tenemos un entorno de desarrollo idóneo para la realización del proyecto.

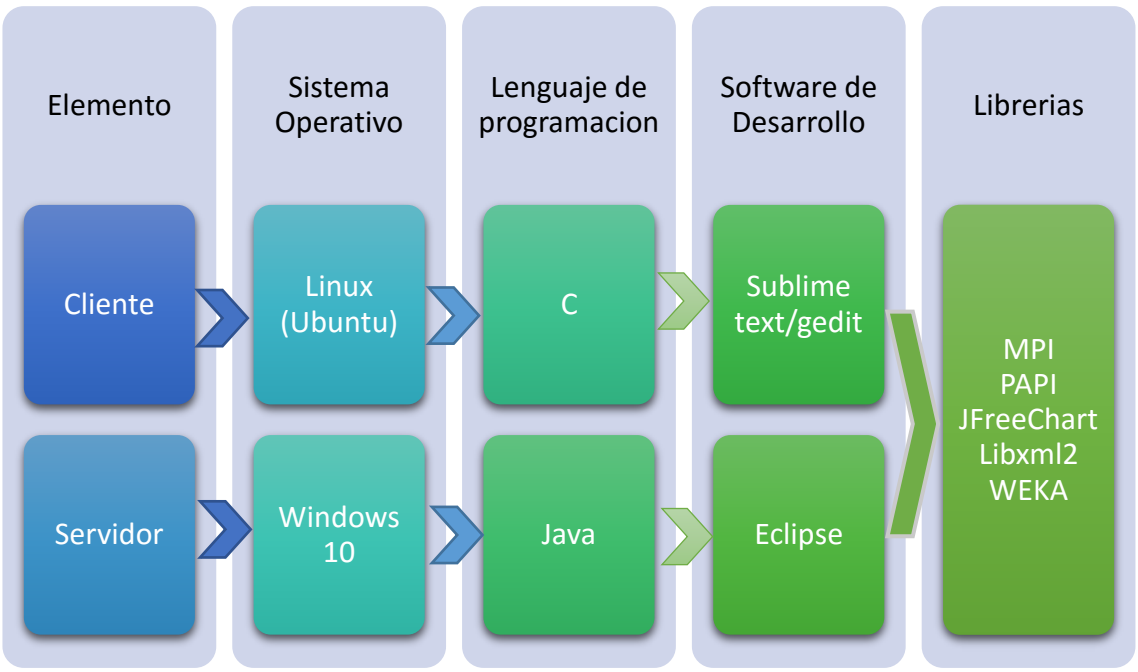

Ilustración 10. Diagrama de configuración del entorno de desarrollo.

4. Arquitectura propuesta

En este apartado se especificará la arquitectura del sistema mediante una serie de diagramas de flujo de datos. Antes de especificar la arquitectura de este sistema, describiré brevemente la arquitectura original, en la cual se ha basado la creación de este sistema.

4.1. Arquitectura original

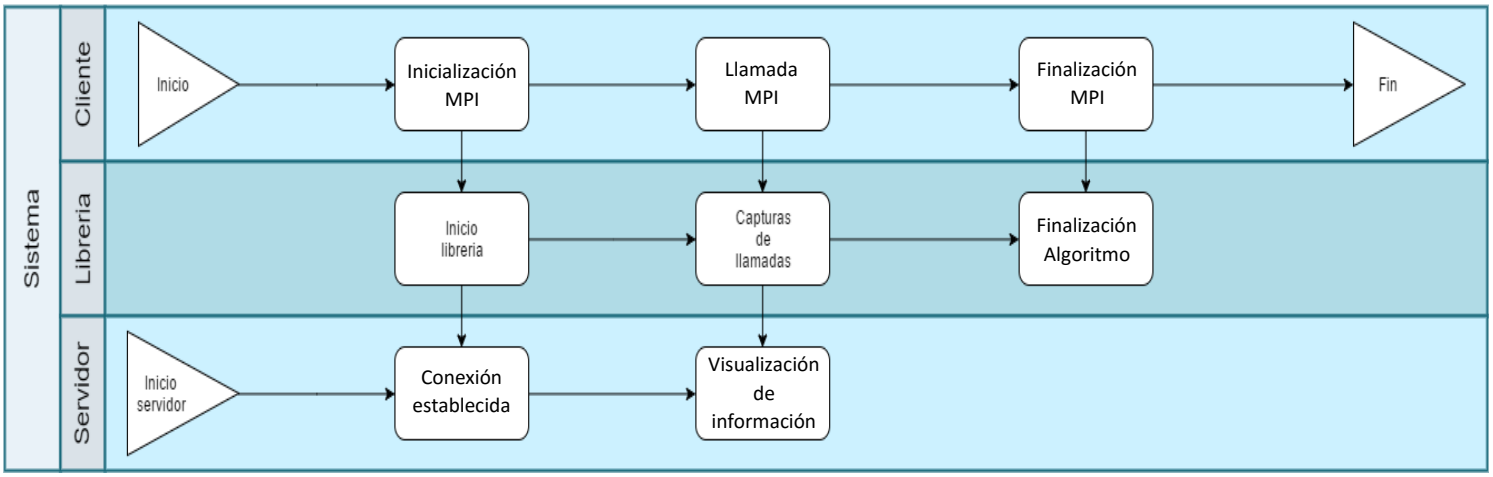

Ilustración 11. Diagrama de arquitectura del sistema original.

Como podemos observar, el sistema está compuesto de tres bloques:

- **Cliente:** la aplicación a monitorizar.
- **Librería:** captura las llamadas realizadas por el cliente.
- **Servidor**: recibe los datos de esa librería y los muestra al usuario.

La aplicación, una vez iniciada, debe seguir el patrón que podemos observar en el diagrama: primero, debe realizar una llamada de inicialización a la librería MPI modificada; a la cual se realizarán llamadas de manera continua durante la ejecución del código de la aplicación; y, al finalizar la ejecución de este, se ha de mandar una llamada de finalización a la librería.

Todas estas llamadas serán capturadas por la librería: cuando se inicializa, la librería MPI realizará una llamada de inicialización a la librería MPI original. A continuación, la librería MPI modificada irá capturando todas las llamadas realizadas por la aplicación a monitorizar hasta que recibe una llamada de finalización, cortando comunicación entre aplicación y librería.

Paralelamente, el servidor estará ejecutándose de manera independiente. Este servidor se mantendrá en espera hasta que la librería le envíe una petición de conexión; y en ese momento inicia un hilo en el que tratará la recepción de información por la librería para ser mostrada y analizada mediante la interfaz.

4.1.1. Librería MPI original

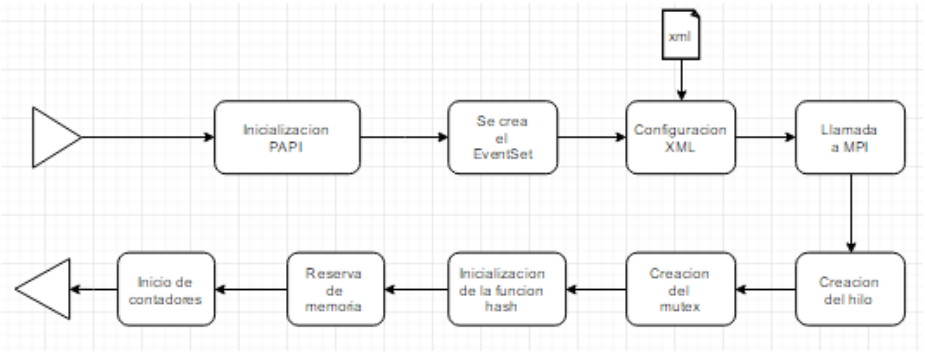

Ilustración 12. Diagrama de arquitectura de la librería MPI original - inicialización.

Como se puede observar, para inicializar la librería se deben cargar los eventos disponibles de PAPI – gracias a la creación del *EventSet* -, y el fichero XML, ya que contiene la configuración de los eventos a analizar y otra información de la conexión – se especificará más en la descripción de la arquitectura actual -. Realizado esto, se procede a hacer la llamada a MPI y a crear el hilo que se encargara de establecer la conexión con la información obtenida del fichero XML – cuya funcionalidad se especificara más detalladamente en el apartado "Arquitectura Actual" - y detectar los periodos; junto a la creación del *mutex* que ordenará el acceso al búfer, evitando así problemas de coherencia. Se inicializa, a continuación, **la función *hash*** de manera que sea aleatoria en cada ejecución del algoritmo, y se reserva la memoria para los búferes de memoria. Finalmente, se inician los contadores PAPI para su posterior lectura.

Cuando se recibe una llamada por parte de la aplicación se captura y se realiza una llamada a la librería modificada que, además de devolver el mismo resultado que la original, recoge información sobre el rendimiento de esa llamada.

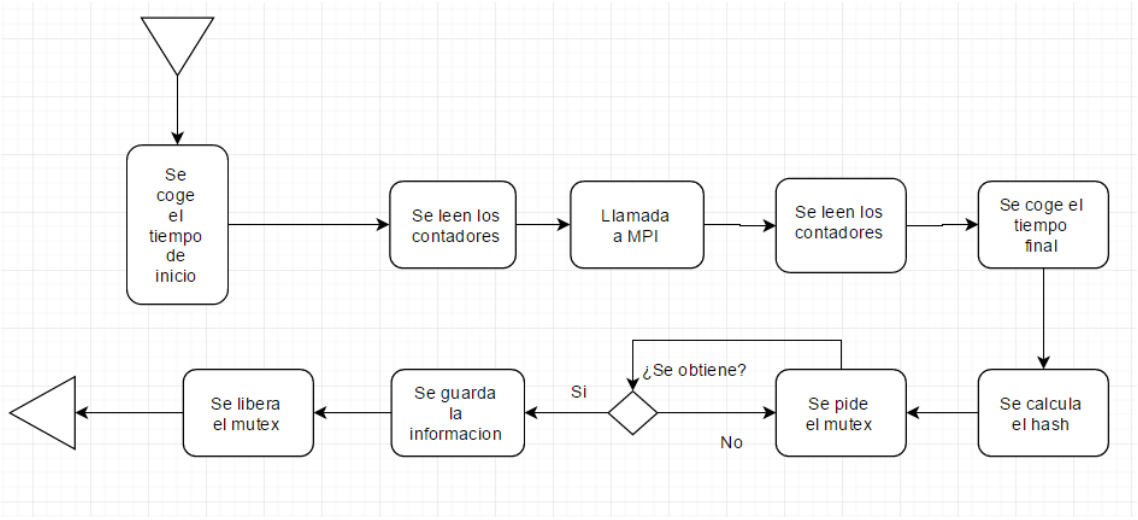

Ilustración 13. Diagrama de arquitectura de la librería MPI original – lectura de contadores.

Para calcular los contadores, se tomará el valor de estos, junto al tiempo, antes de realizar la llamada original a MPI, y luego después de ejecutarse; siendo la diferencia entre esos dos valores el realizado para la llamada MPI ejecutada. A continuación se calcula su *hash* para reconocer la misma llamada en caso de repetirse y, una vez procesada, se accede el *mutex* para acceder al búfer y guardar el hash, el tiempo y los contadores PAPI.

Finalmente, para finalizar la librería, se destruirá el *mutex* y se finalizara el hilo; además de realizar una llamada de parada a la librería original MPI y se paran los contadores.

4.1.2. Servidor original

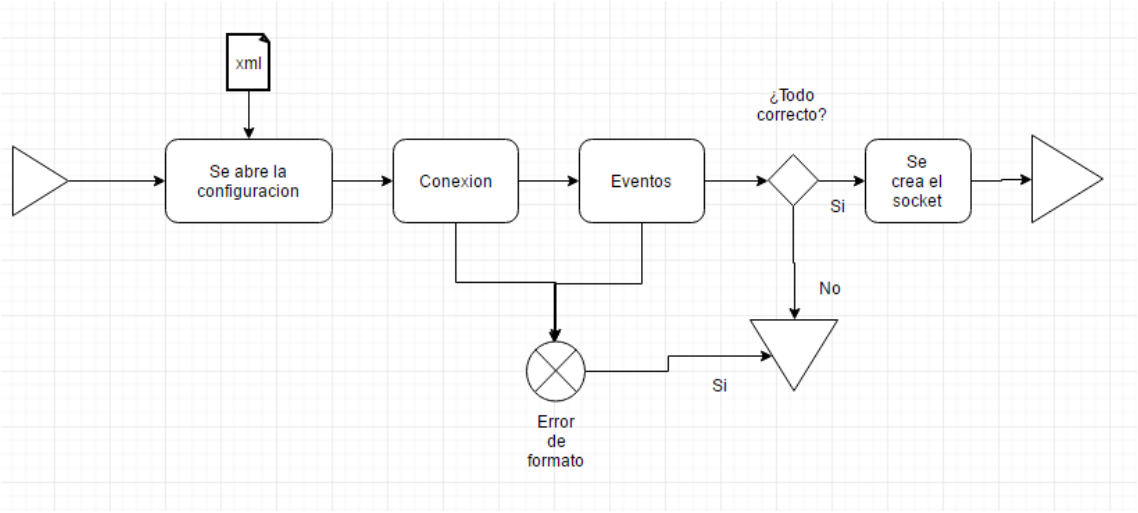

Ilustración 14. Diagrama de arquitectura del servidor original - inicialización.

El primer paso a tomar con el servidor es inicializarlo. Para ello, tenemos que abrir la configuración del fichero XML. Tras comprobarlo, se crea el socket en el puerto especificado.

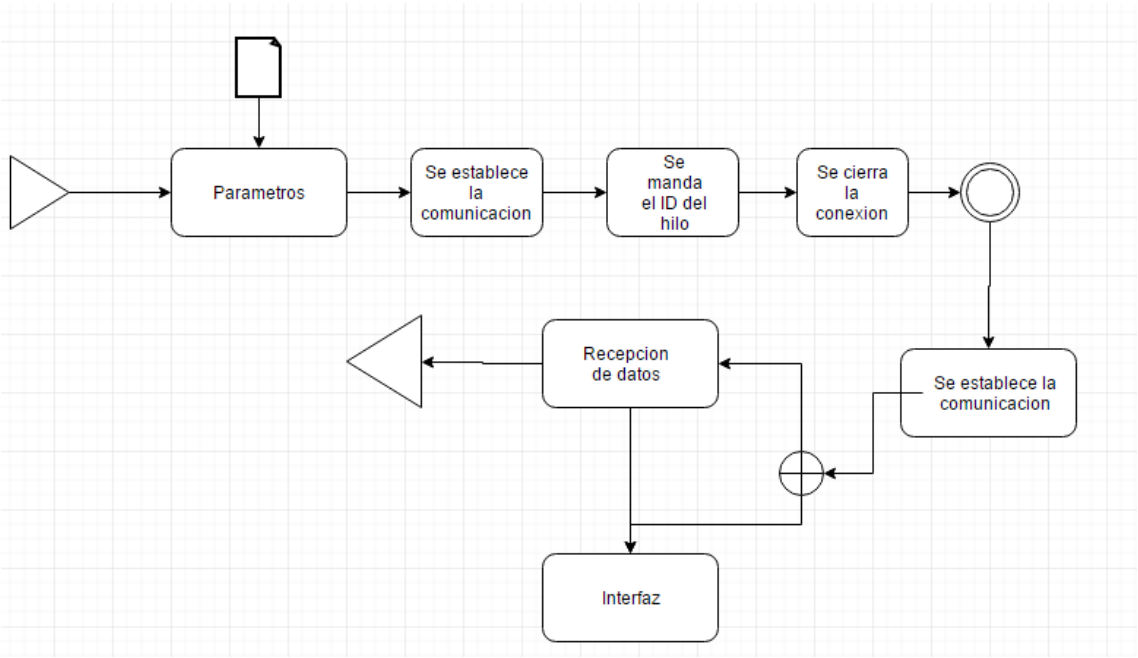

Ilustración 15. Diagrama de arquitectura del servidor original – funcionamiento general.

Cada vez que se recibe una petición de conexión, se crea un hilo, asociado a un puerto y se manda la ID del hilo, cerrando esa conexión. A continuación se crea la interfaz gráfica y lee la información que recibe del cliente; mientras que actualiza la interfaz con esta información a tiempo real.

4.2. Arquitectura actual

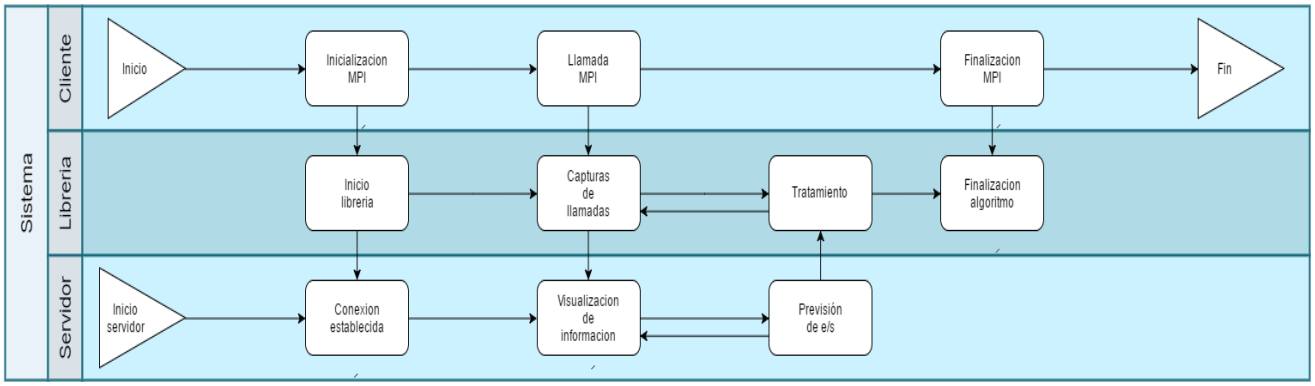

Ilustración 16. Diagrama de arquitectura del sistéma actual.

La arquitectura actual realizada es muy similar a la original. El sistema se compone, también, de tres bloques, a los cuales se les han añadido nuevas funcionalidades para cumplir con los objetivos establecidos:

- El **cliente** seguirá realizando llamadas MPI, pero se le ha incluido algunas llamadas de entrada/salida para su detección y posterior análisis.
- La **librería** se inicializará cuando reciba la llamada de inicialización del cliente. Esta función mejorada de la librería seguirá capturando las llamadas realizadas por el cliente y se las mandará al servidor, pero además ahora recibirá información del servidor y realizará un tratamiento dependiendo de lo recibido.
- El **servidor** se iniciará en paralelo a la vez que el cliente, y se conectará a la librería. Esta versión mejorada del servidor establecerá una comunicación bidireccional con la librería: recibirá información de los contadores desde ella, y le enviará las acciones que el cliente deberá seguir.

Una vez finalizadas todas esas funcionalidades, el cliente le envía una llamada de finalización a la librería, y se acaba la ejecución del sistema.

A continuación se detallarán las nuevas funcionalidades, y su papel dentro del sistema.

4.2.1. Librería MPI modificada

Como hemos especificado antes, la funcionalidad original de la librería es la de capturar los eventos y mandárselas al servidor. Esta funcionalidad sigue siendo la principal en nuestro sistema, pero hemos añadido una nueva funcionalidad al sistema: el tratamiento de los clientes en ejecución, el cual dependerá de la información recibida del servidor.

Inicialización de la librería

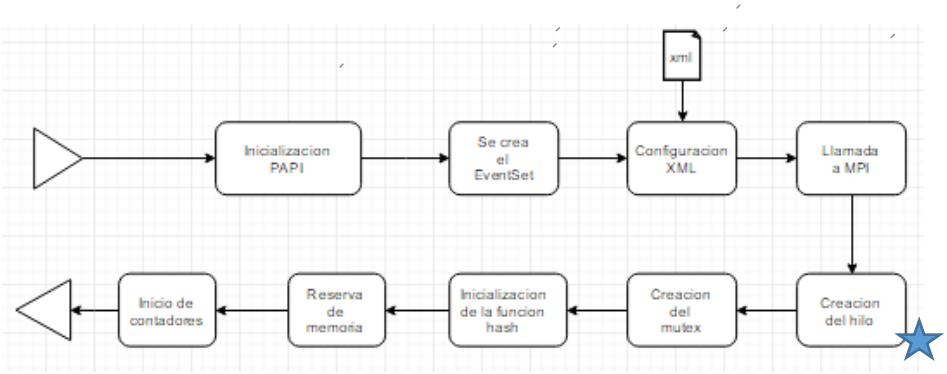

Ilustración 17. Diagrama de arquitectura de la librería actual - inicialización.

Como podemos observar, se realiza el mismo patrón que la original: se inicializa PAPI, se crea el *EventSet* y se carga la configuración del archivo XML, el cual contiene las siguientes etiquetas:

- ***Conexión***: parámetros para efectuar la conexión, con que puerto y que host.
- ***PAPI***: permite elegir el número y tipo de eventos o contadores van a leer. En la versión modificada, al menos uno de ellos ha de ser PAPI_FPS_OPS. Este contador, cuando se realiza un evento de entrada/salida contabiliza muchas menos operaciones, ya que en estos eventos no se usan; siendo muy útil para la detección de este tipo de eventos.
- ***Algoritmo***: parámetros útiles para la detección de los periodos, así como para el reinicio del algoritmo o si se tiene que retrasar la ejecución del hilo.
- ***Búfer***: parámetros que indicarán el tamaño del búfer que almacenará y la matriz que enviará al servidor.

Tras haber leído esta configuración y haber establecido los parámetros de configuración en el programa, se realiza la llamada al MPI original, inicializando esta librería. A continuación se crea el hilo y se ejecutará de una manera distinta al sistema original. Al ser una tarea más compleja, se especificará en el siguiente sub-apartado su ejecución. Finalmente, se crea el *mutex*, se inicializara la función hash y la reserva de memoria y se iniciarán los contadores; justo de la misma manera que en el sistema original.

Creación y ejecución del hilo.

Como hemos dicho anteriormente, la creación y ejecución requiere de distintas etapas, ya que contiene la mayor parte del núcleo de ejecución del hilo.

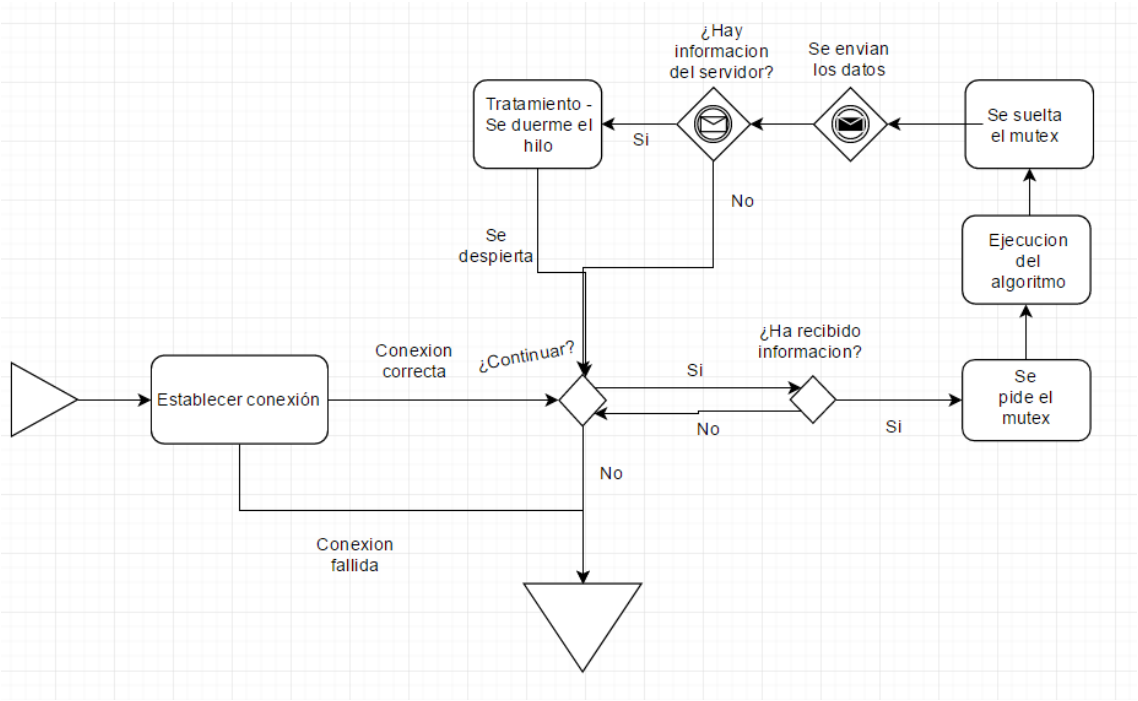

Ilustración 18. Diagrama de arquitectura del servidor actual – creación y ejecución del hilo.

El primer paso es establecer conexión con el servidor. Esto será posible gracias a los parámetros que hemos introducido en el apartado "Conexión" del fichero XML; ya que se enviará la petición de conexión al host indicado, y al puerto establecido por defecto. Si ha sido satisfactorio, se recibirá un puerto nuevo, y se cerrará el socket; creando uno nuevo hacia el puerto establecido por esta llamada. Si los anteriores pasos han sido satisfactorios, comenzará con la ejecución del hilo, la cual se puede dividir en dos bloques:

Procesamiento de datos.

Este bloque se ejecuta tal y como se ejecutaba en el sistema original: si la ejecución tiene indicado que continúe, comprobará si hay datos para leer. Si no es así, esperará hasta tener datos. Una vez obtenidos los datos, pedirá el *mutex* para acceder al búfer de llamadas junto con los contadores y, cuando ese permiso sea concedido, se cargarán esos datos y se enviarán al algoritmo de detección de periodos. Cuando todos estos datos estén procesados, se liberará el *mutex* y se enviarán las matrices al servidor.

Tratamiento de resultados

Este bloque se ejecutará a continuación de enviar los datos al servidor. Tras procesar el servidor los resultados obtenidos por el bloque anterior, enviará a la librería un mensaje con el tratamiento a tomar. En el caso del proyecto, este tratamiento es mantener en espera a la aplicación hasta que se termine el uso de entrada/salida de otra aplicación, por lo que enviará un mensaje con un tiempo de espera, el cual deberá estar esta aplicación suspendida. Cuando sale de ese estado de suspensión, seguirá ejecutándose con normalidad.

Resto de funcionalidades

El resto de funcionalidades realizan las mismas tareas que en el original:

- Las capturas de llamadas, como especificamos antes, se obtendrá mediante la captura de las llamadas MPI, y el cálculo del tiempo y de los contadores anterior y posterior a la llamada MPI, y se guardará la información en el búfer junto a su hash.
- La finalización de la librería se realizará, también, con la captura de la llamada de finalización a MPI, que se pasará a la librería original y a su vez se destruirá tanto el *mutex* como el hilo, además de detener los contadores PAPI.

4.2.2. Conexiones cliente-servidor

En este apartado se especificará la conexión cliente – servidor, así como su comparación con la original y su modo de ejecución tanto en el cliente como en el servidor.

Diferencias con la original

En el sistema base disponíamos de una conexión como la representada en el siguiente diagrama:

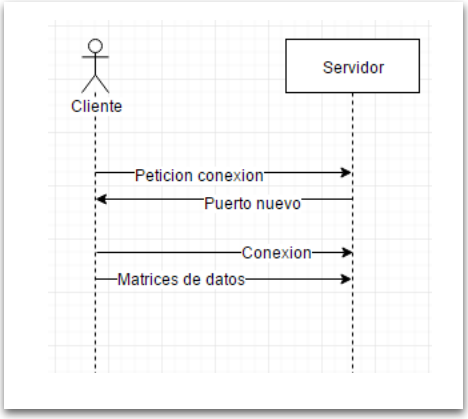

Ilustración 19. Diagrama de la conexión original.

Como se ve, al principio es un sistema bidireccional, para establecer los parámetros de conexión (*petición + establecimiento de puerto*); pero, a la hora de enviar datos, es unidireccional – únicamente el cliente le envía la información al servidor.

El sistema actual necesita que esta comunicación sea bidireccional, ya que el servidor ya no solo recibe los datos, si no que decide que tratamiento dar al cliente, y tiene que enviarle este tratamiento – en forma de tiempo de espera – al usuario, por lo que necesitamos una conexión como la siguiente:

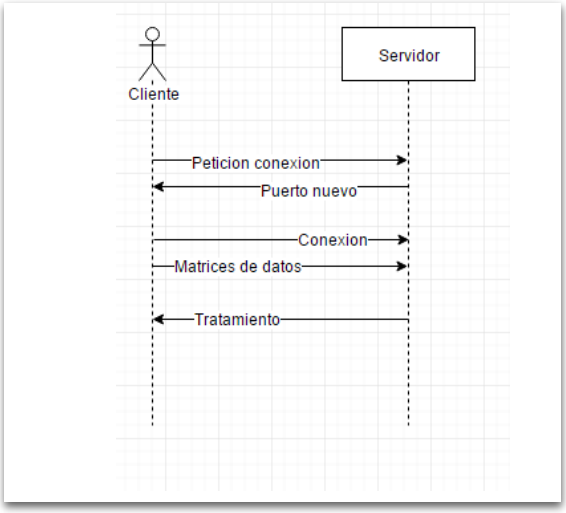

Ilustración 20. Diagrama de la conexión actual.

Estas conexiones se establecerán según los patrones que definiremos a continuación.

Establecimiento de conexión del cliente

La tarea se detalla en el siguiente diagrama:

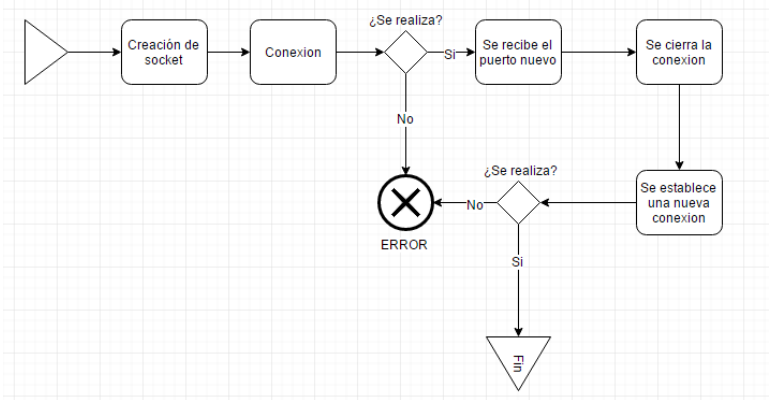

Ilustración 21. Diagrama del establecimiento de conexión del cliente actual.

Se crea el *socket* con la información al puerto por defecto y host determinado, entonces recibe el nuevo puerto, al que enviará toda la información de los contadores y se cierra la conexión con el puerto antiguo. Se vuelve a abrir una conexión nueva con el puerto nuevo y, si ha ido todo bien, se continúa con la ejecución del programa.

Establecimiento de conexión del servidor

La tarea se detalla en el siguiente diagrama:

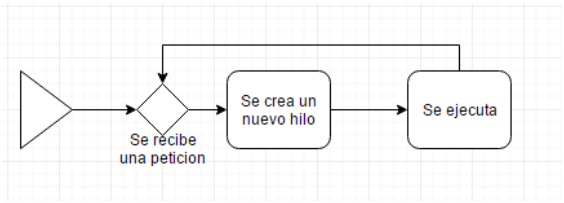

Ilustración 22. Diagrama del establecimiento de conexión del servidor actual.

El servidor recibe una petición, y crea un nuevo hilo pasando por parámetro la ID de este, para la definición del puerto que se le va asignar. Se envía esta información al cliente y se comienza la ejecución del hilo, la cual especificaremos a continuación.

4.2.3. Servidor actual

En el caso del bloque servidor, ha pasado de tener un papel totalmente observador y pasivo a ser el bloque activo del sistema. En el sistema original, el servidor únicamente recibe información y la muestra. En el sistema actual, además de eso, detecta y predice eventos de entrada y salida, e informa a la librería de cómo debe de tratar al cliente a partir de estos datos. En el sistema actual este tratamiento es un mensaje con el valor del tiempo que debe mantenerse en espera el cliente en caso de colisión para acceder a un recurso, en este caso, la CPU.

A continuación se describirán las funcionalidades del servidor.

Inicialización del servidor

En el caso de la inicialización del servidor, se siguen los mismos pasos que en el original, como podemos ver en el siguiente diagrama.

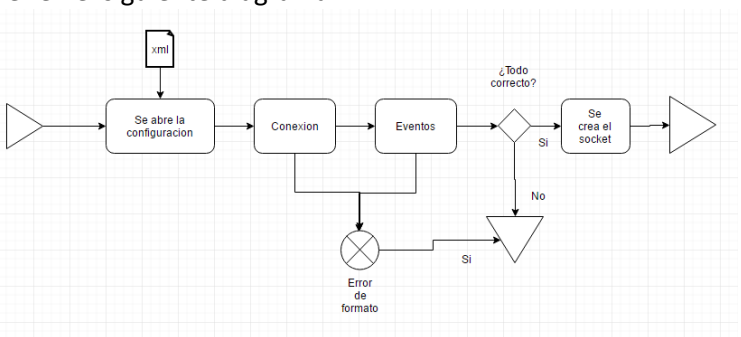

Ilustración 23. Diagrama de la arquitectura del servidor actual - inicialización.

El primer paso a tomar con el servidor es inicializarlo. Para ello, tenemos que abrir la configuración del fichero XML.

- **Conexión**: el puerto por defecto cada vez que se intente conectar un cliente.
- **PAPI**: los eventos que se monitorizarán, usado en las etiquetas de la interfaz.
- **Búfer**: parámetros de tamaño del búfer.

Tras comprobar que todos esos parámetros son válidos, y establecerlos en la ejecución, se crea el socket en el puerto especificado.

Ejecución del hilo

En este bloque se encuentran todas las modificaciones del servidor. Para ello, se mostrará primero el diagrama de ejecución, con más pasos que el original:

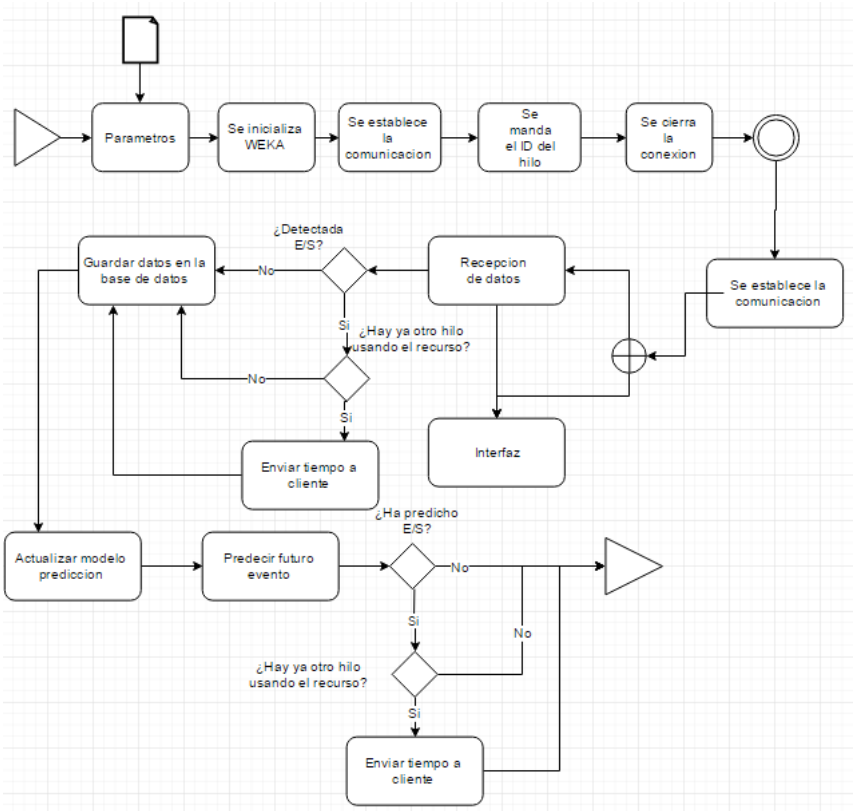

Ilustración 24. Diagrama de la arquitectura del servidor actual – ejecución del hilo.

Como podemos observar, el comienzo es similar al sistema original. Es, a partir de la recepción de datos, cuando se complica la arquitectura. Cuando se recibe los datos, a partir de los contadores, se detecta si hay un evento de E/S; es decir, cuando se recibe un pico de bajada en el contador ***PAPI_FP_OPS***, se detectan eventos de E/S y, cuando se detecta un pico de subida, se detecta que se ha salido de esos eventos. Esto es debido a que, cuando un proceso está realizando instrucciones de E/S, está escribiendo en disco y no está haciendo uso de la CPU, por lo que los FLOPS no están siendo utilizados, y por eso baja el número de estos.

A continuación, se sigue el siguiente tratamiento:

- Si ha detectado eventos de E/S…
  - … y no hay otro hilo usando el recurso, se hace comunica al servidor que hay un hilo usando el recurso. Esto se realiza mediante un *flag* global en el servidor al cual se accede mediante *mutex*, para evitar fallos de coherencia.
  - … y hay otro hilo usando el recurso, comunica al servidor que hay un hilo más esperando para el recurso y se calcula el tiempo de parada, el cual se envía al cliente. El número de hilos en espera es también una variable global a la que se accede mediante *mutex*.
- Si no ha detectado eventos de E/S reanuda con su ejecución normal.

Después de realizar esta primera comprobación, el servidor guardará los datos del valor del contador, el tiempo desde los últimos eventos, y el estado en el que se encuentra la aplicación; en un archivo *.arff*, de base de datos de WEKA, del cual se nutrirá el modelo de datos. Este modelo, basado en *Machine Learning*, toma esta base de datos y crea un modelo, en el caso de nuestro sistema, un *Random Tree*, el cual usará el servidor para predecir los eventos.

A continuación se predecirá, en base al modelo de datos anteriormente descrito, si se realizan algunos eventos de E/S en el futuro, con una holgura de predicción de 2 ciclos. A partir de esta predicción se seguirá la misma decisión que cuando se detectaba un evento de E/S.

Machine Learning

Como se ha especificado antes, en algoritmo utilizado en la predicción de eventos ha sido un algoritmo de *Random Tree* (Aldous, 1991) . Este algoritmo se trata de un árbol de decisión, es decir, se mapean observaciones sobre los contadores, para ofrecer una predicción basada en los datos recogidos.

Se ha escogido este algoritmo por dos principales motivos:

- Es ideal si las alternativas están bien definidas – en nuestro caso, alternativa binaria – y si las incertidumbres pueden ser cuantificadas – en nuestro caso, la probabilidad que sea E/S o no.
- Tras un análisis en la herramienta gráfica WEKA y experimentación con los datos en diferentes algoritmos, este fue el que menos error en la predicción obtuvo.

Este algoritmo se nutre de los siguientes elementos para la predicción de eventos:

- ***Instrucciones***: las instrucciones en el momento de toma de medidas. Lo usa el algoritmo, secundariamente, para crear ramas del árbol.
- ***Tiempo***: tiempo desde la última E/S. Es usado, principalmente, por el algoritmo para crear ramas del árbol, debido a la periodicidad de los eventos E/S.
- ***Flag***: estado en el que se encuentra el sistema en el momento de la medida. Se usa como incógnita para predecir, ya que se quiere saber su valor en determinado tiempo.

4.3. Análisis de requisitos

En este apartado se especificaran los requisitos de nuestro sistema, los cuales definirán tanto el funcionamiento como las restricciones del sistema, siempre cumpliendo con los objetivos especificados.

Estos requisitos han sido creados, revisados y modificados a lo largo del proyecto, adecuándose a las decisiones tomadas y los cambios realizados.

Seguiremos el patrón especificado durante la carrera, siendo la plantilla usada la siguiente:

Identificador: RS-XXXX			
Titulo		Fecha	XX/XX/XXXX
Descripción			
Necesidad	☐ Esencial ☐ Deseable ☐ Opcional	Prioridad	☐ Alta ☐ Media ☐ Baja
Verificabilidad	☐ Alta ☐ Media ☐ Baja	Estabilidad	☐ Estable ☐ Inestable
Relación/es			

Tabla 2. Patrón de requisito.

Los elementos que componen cada requisito son:

- **Identificador**: código único que identifica al requisito. Se compondrá de los caracteres RS (*Requisito del Sistema*) + 4 caracteres *(2 que indiquen el tipo de requisito + 2 que indiquen numéricamente su posición en esa categorización*).
- **Título:** título descriptivo del requisito.
- **Fecha:** fecha en la que fue realizada la última versión del requisito.
- **Descripción**: pequeña definición del requisito.
- **Necesidad**: indica la relevancia en la funcionalidad del proyecto.
    - ***Necesario*:** imprescindible.
    - ***Deseable***: afecta considerablemente al proyecto.
    - ***Opcional*:** solamente incluye una nueva característica.
- **Prioridad:** la importancia que tiene el requisito dentro del proyecto.
- **Verificabilidad**: Indica el nivel de comprobación posterior al desarrollo.
- **Estabilidad**: indica la inestabilidad a lo largo del desarrollo.
- **Relación**: indica las posibles dependencias que tiene respecto a otros requisitos.

4.3.1. Requisitos Funcionales

Estos requisitos especificaran la funcionalidad o los servicios que debe proporcionar nuestra aplicación. Su etiqueta de requisito es FU.

Identificador: RS-FU01			
Titulo	Compatibilidad con librerías.	Fecha	06/09/2016
Descripción	El sistema deberá ser compatible con las librerías MPI, PAPI y WEKA.		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es			

Tabla 3. Requisito RS-FU01.

Identificador: RS-FU02			
Titulo	Contadores Hardware.	Fecha	06/09/2016
Descripción	El sistema dará información sobre los contadores hardware.		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es	RS-FU01		

Tabla 4. Requisito RS-FU02.

Identificador: RS-FU03			
Titulo	Detección de eventos E/S.	Fecha	06/09/2016
Descripción	El sistema detectará cuando se está realizando un evento de entrada/salida.		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es	RS-FU02, RS-FU01		

Tabla 5. Requisito RS-FU03.

Identificador: RS-FU04			
Titulo	Predicción de eventos E/S.	Fecha	06/09/2016
Descripción	El sistema predecirá, en la mayor medida de lo posible, los eventos de entrada/salida.		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es	RS-FU02, RS-FU03, RS-FU01		

Tabla 6. Requisito RS-FU04.

Identificador: RS-FU05			
Titulo	Paralelización.	Fecha	06/09/2016
Descripción	El sistema se deberá ejecutar paralelamente a la aplicación que analiza.		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es			

Tabla 7. Requisito RS-FU05.

Identificador: RS-FU06			
Titulo	Calculo eventos hardware.	Fecha	06/09/2016
Descripción	El sistema calculará el incremento en contadores PAPI para cada llamada que se realice a MPI.		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es	RS-FU01, RS-FU02		

Tabla 8. Requisito RS-FU06.

Identificador: RS-FU07			
Titulo	Servidor.	Fecha	06/09/2016
Descripción	El sistema se ejecutará en un servidor.		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es			

Tabla 9 Requisito RS-FU07.

Identificador: RS-FU08			
Titulo	Ventanas.	Fecha	06/09/2016
Descripción	El sistema abrirá una ventana de interfaz por cada hilo del proceso.		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es	RS-FU05, RS-FU16		

Tabla 10. Requisito RS-FU08.

Identificador: RS-FU09			
Titulo	Comunicación cliente-servidor.	Fecha	06/09/2016
Descripción	El sistema permitirá una comunicación bidireccional entre el programa y las aplicaciones que se conectan a él.		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es	RS-FU01		

Tabla 11. Requisito RS-FU09.

Identificador: RS-FU10			
Titulo	Parada de proceso.	Fecha	06/09/2016
Descripción	El sistema detendrá el proceso cuando vea oportuno, según se ha especificado en la arquitectura.		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es	RS-FU09, RS-FU01, RS-FU02		

Tabla 12. Requisito RS-FU10.

Identificador: RS-FU11			
Titulo	Configuración servidor.	Fecha	06/09/2016
Descripción	El sistema dispondrá de un archivo XML editable a los usuarios. Este fichero contendrá los siguientes datos: • Tamaño de los búfer empleados. • Prioridad de ejecución del hilo. • Periodo máximo y reinicio del algoritmo de detección. • Conexión con el servidor. • Contadores PAPI.		
Necesidad	☐ Esencial ☐ Deseable ☒ Opcional	Prioridad	☐ Alta ☒ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es	RS-FU01		

Tabla 13. Requisito RS-FU11.

Identificador: RS-FU12			
Titulo	Creación de fichero de datos.	Fecha	06/09/2016
Descripción	El sistema creará un fichero de datos arff con los datos obtenidos de la recogida de los contadores. Este fichero contendrá los siguientes datos: • Valor del contador • Tiempo • Estado		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es	RS-FU01		

Tabla 14. Requisito RS-FU12.

Identificador: RS-FU13			
Titulo	Lectura de fichero de datos.	Fecha	06/09/2016
Descripción	El sistema podrá leer información de los ficheros .arff.		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es	RS-FU12, RS-FU01		

Tabla 15. Requisito RS-FU13.

Identificador: RS-FU14			
Titulo	Machine Learning.	Fecha	06/09/2016
Descripción	El sistema usará algoritmos y funciones de Machine Learning para predecir futuros datos.		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es	RS-FU01		

Tabla 16. Requisito RS-FU14.

Identificador: RS-FU15			
Titulo	Feedback.	Fecha	06/09/2016
Descripción	El sistema informará a los usuarios cuando ha cambiado de estado cada hilo mediante un aviso en el terminal.		
Necesidad	☐ Esencial ☒ Deseable ☐ Opcional	Prioridad	☐ Alta ☒ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es			

Tabla 17. Requisito RS-FU15.

Identificador: RS-FU16			
Titulo	Interfaz.	Fecha	06/09/2016
Descripción	El sistema mostrará la información de los contadores mediante una interfaz gráfica.		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es	RS-FU01, RS-FU02		

Tabla 18. Requisito RS-FU16.

4.3.2. Requisitos No Funcionales.

Los requisitos no funcionales, o de restricción, son restricciones que afectaran al funcionamiento del sistema. Su etiqueta de requisito será NF.

Identificador: RS-NF01			
Titulo	Requisitos hardware.	Fecha	06/09/2016
Descripción	El sistema se usará en los siguientes sistemas: • iMedia i5-650/3.20 GHz • Toshiba Satellite i3-3110/2.4 GHz		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es			

Tabla 19. Requisito RS-NF01.

Identificador: RS-NF02			
Titulo	Sistemas Operativos.	Fecha	06/09/2016
Descripción	El sistema se ejecutará en los siguientes sistemas operativos: • Ubuntu Linux • Windows 10		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es			

Tabla 20. Requisito RS-NF03.

Identificador: RS-NF03			
Titulo	Requisitos software.	Fecha	06/09/2016
Descripción	El sistema se desarrollara en el siguiente software: • Eclipse • Sublime Text 3 • Gedit • MPICH v 1.4.1 • PAPI v5.1 • JCommon v1.0.17 • JFreeChart v1.0.14 • WindowsBuilder • Libxml2 v2.9.1 • WEKA 3		
Necesidad	☒ Esencial ☐ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es			

Tabla 21. Requisito RS-NF03.

Identificador: RS-NF04			
Titulo	Mensajes de error.	Fecha	06/09/2016
Descripción	La aplicación informará a los usuarios de los errores conocidos mediante descripciones de estos fallos.		
Necesidad	☐ Esencial ☒ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es			

Tabla 22. Requisito RS-NF04.

Identificador: RS-NF05			
Titulo	Mensajes de ayuda.	Fecha	06/09/2016
Descripción	La interfaz mostrará ayudas a los usuarios.		
Necesidad	☐ Esencial ☒ Deseable ☐ Opcional	Prioridad	☒ Alta ☐ Media ☐ Baja
Verificabilidad	☒ Alta ☐ Media ☐ Baja	Estabilidad	☒ Estable ☐ Inestable
Relación/es			

Tabla 23. Requisito RS-NF05.

4.4. Diagrama de clases e identificación de clases.

Nuestro sistema está compuesto por clases independientes: servidor, librería MPI e interfaz, las cuales no tienen ni relación específica ni atributos característicos; por lo que se ha descartado realizar un análisis de clases.

4.5. Definición de interfaces de usuario

En este apartado se van a describir como serán las interfaces de usuario de nuestro sistema. Se describirán sus principios generales, sus perfiles y diálogos, su comportamiento dinámico, etc.; buscando con estas características una interfaz útil para el usuario a la vez de satisfacer los objetivos establecidos. De este modo se crearán interfaces de fácil manejo, coherentes, acordes a los objetivos y flexibles.

El sistema se basa en una ventana con tres pantallas, las cuales se describirán a continuación:

4.5.1. Pantalla de matriz de incrementos

Para comentar esta pantalla, adjuntaré una captura de pantalla que completara mi explicación:

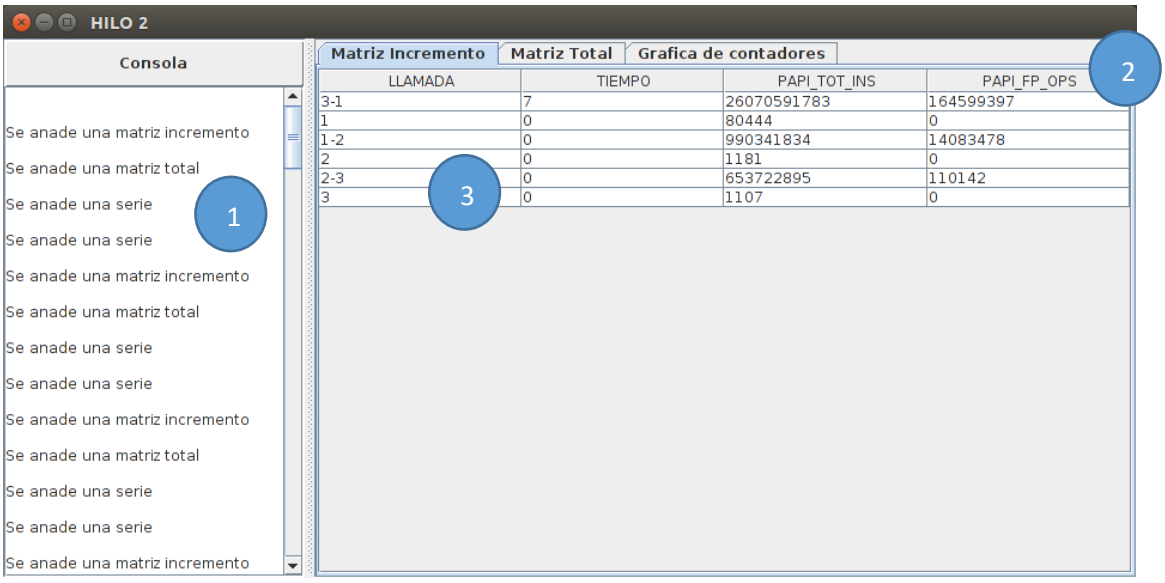

Ilustración 25. Interfaz de matriz de incrementos.

Esta pantalla muestra la última información recibida por cada cliente. En ella podemos ver la matriz de incrementos, en la cual se puede visualizar si hay alguna variación brusca entre los datos recibidos.

Los bloques característicos son los siguientes:

1. Consola que muestra mensajes de información al usuario, así como cambios en el sistema.
2. Identificador de cada columna. Los identificadores de los contadores irán indicados con su nombre especificado en el fichero cabecera de PAPI.
3. Información sobre:
    a. Llamada en la que se está midiendo los datos.
    b. Tiempo de cómputo empleado.
    c. Información de los contadores que se están analizando.

4.5.2. Pantalla de información global

La siguiente pantalla es muy similar a la anterior, la única diferencia es la información mostrada:

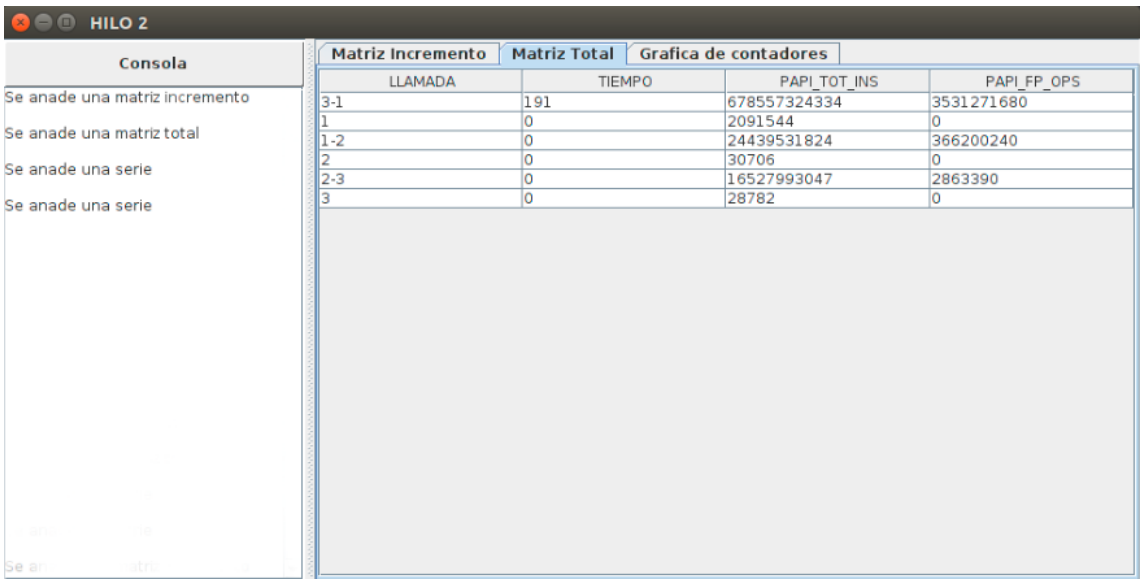

Ilustración 26. Interfaz de información global.

En esta pantalla se muestra la acumulación total de los valores de cada fila en la parte de código especificada en la columna de llamada. Esto sirve para poder observar que parte del código es la que más carga posee dentro del sistema en total.

4.5.3. Pantalla de grafico

A continuación se adjuntará la pantalla del gráfico:

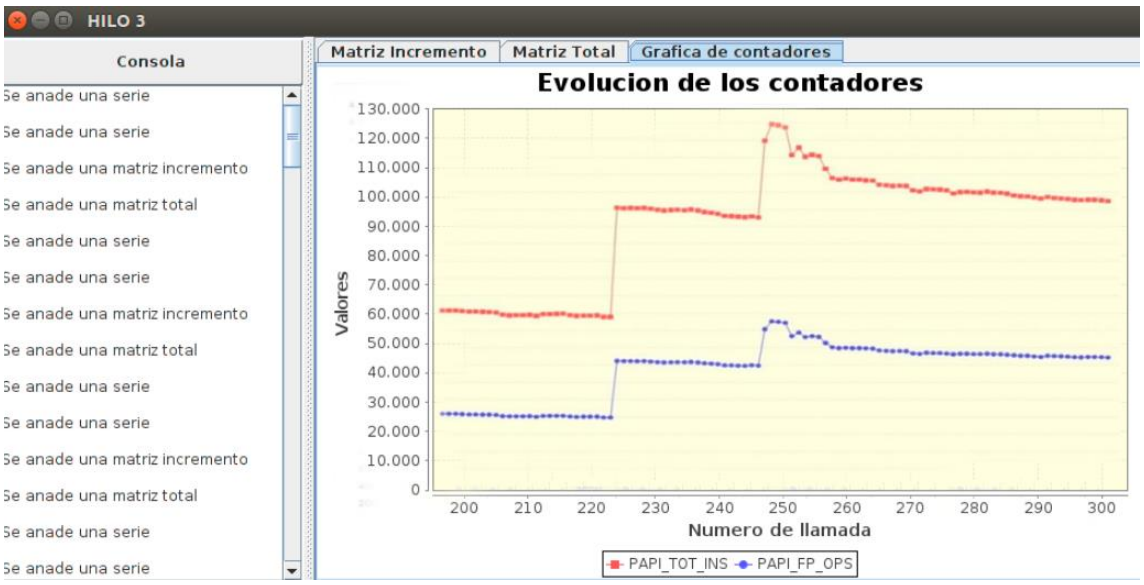

Ilustración 27. Pantalla de gráficas de contadores.

En ella se pueden ver todos los valores de los contadores en las últimas llamadas, en forma de gráfica, para que pueda comprobar el rendimiento del proceso; ya que se puede observar cuando ha habido variaciones o picos de valores, y en que llamada, para poder analizarla y encontrar el motivo de esos picos. Además, incluye una consola en la que también se muestran los mensajes relacionados con este rendimiento.

5. Planificación

En este apartado se especificaran las estimaciones de planificación temporal correspondientes al proyecto, además de una estimación de costes acerca de los recursos, tanto hardware como software, utilizados en el proyecto. Cabe destacar la variabilidad de estas estimaciones a lo largo del proyecto.

5.1. Planificación temporal

La planificación temporal se realizará mediante diagramas de Gantt (Gantt, 1910), en los que se realizará un diagrama que ubicarán las tareas a realizar en el tiempo del proyecto.

5.1.1. Planificación global

A continuación se mostrará la planificación global del proyecto, incluyendo la duración en días de cada apartado. El proyecto tiene una duración de 127 días, del 21 de Mayo al 24 de Septiembre. Además, se mostrará el diagrama de Gantt global, con los apartados, los cuales se explicarán más adelante y se mostrarán sus respectivos diagramas.

- Planificación: 17 días.
- Estudio del sistema base: 23 días.
- Estudio de viabilidad del sistema: 8 días.
- Análisis del sistema: 18 días.
- Diseño del sistema: 11 días.
- Implementación del sistema: 30 días.
- Pruebas: 20 días.

Su representación en el diagrama de Gantt es la siguiente:

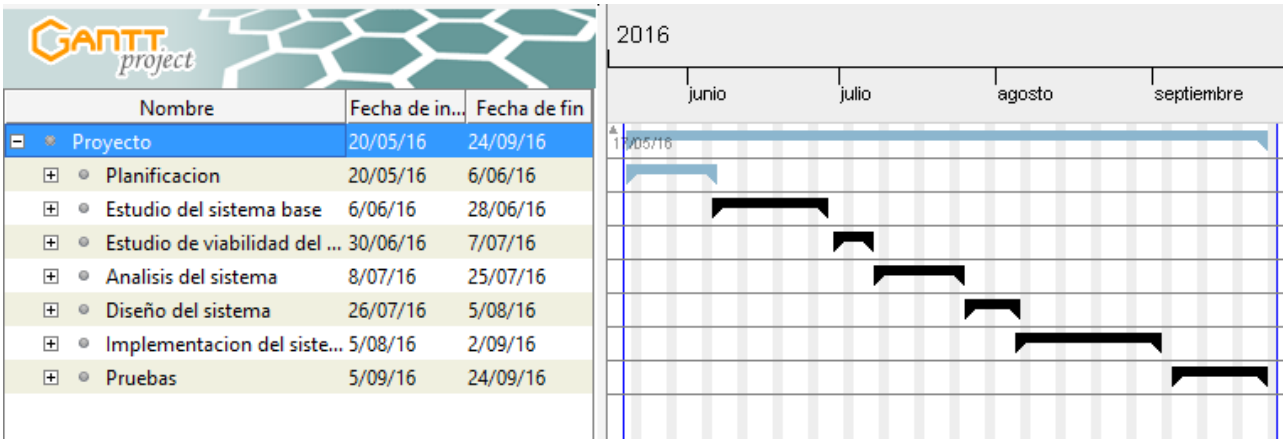

Ilustración 28. Diagrama de Gantt global.

5.1.2. Planificación

La planificación del proyecto abarca tanto las reuniones iniciales con el tutor como el estudio de la idea del proyecto y la documentación sobre las librerías y los lenguajes utilizados.

A continuación se muestra tanto los componentes del apartado como su diagrama de Gantt:

- Planificación: 17 días.
    - Reuniones con el tutor: 2 días.
    - Lectura de la idea: 1 día.
    - Lectura de documentación: 3 días.
    - Estudio de librería PAPI: 2 días.
    - Estudio de librerías Machine Learning: 2 días.
    - Estudio de librería WEKA: 1 día.
    - Estudio de librería MPI: 2 días.
    - Elaboración de la documentación: 4 días.

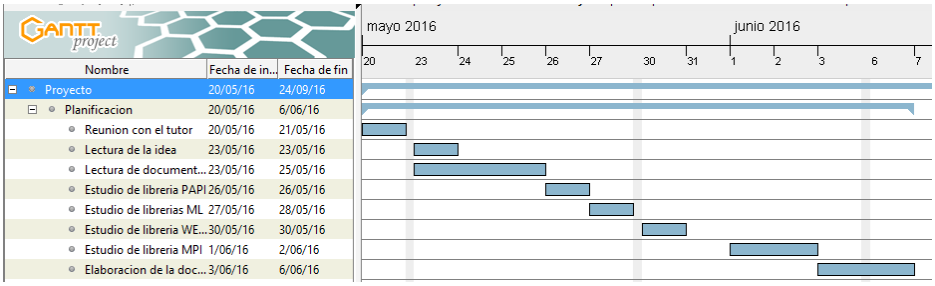

Ilustración 29. Diagrama de Gantt - planificación.

5.1.3. Estudio del sistema base

El estudio que se ha realizado sobre el sistema base es la comprensión tanto de la memoria como del código del sistema en el que se ha basado el actual. Los componentes de este apartado serán los siguientes:

- Estudio del sistema base: 23 días.
    - Reunión con el tutor: 2 días.
    - Lectura de la memoria base: 5 días.
    - Comprensión del código base: 7 días.
    - Pruebas del código base: 5 días.
    - Elaboración de la documentación: 4 días.

Y su representación en diagrama es el siguiente:

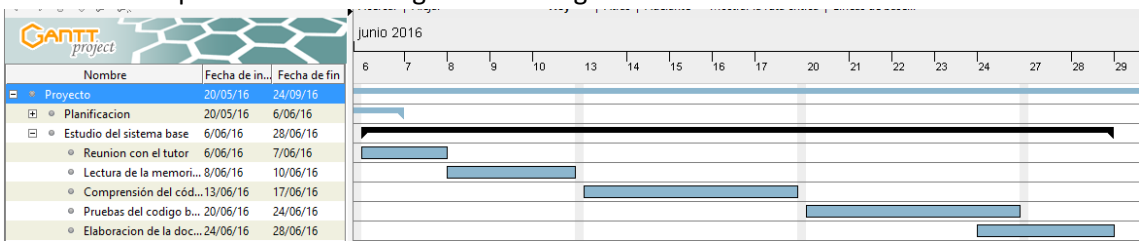

Ilustración 30. Diagrama de Gantt – estudio del sistema base.

5.1.4. Estudio de viabilidad del sistema

En esta fase se realizará un estudio de las herramientas similares a nuestro sistema en el mercado actual. Esta fase se compondrá de los siguientes elementos:

- Estudio de viabilidad del sistema: 8 días.
    - Estudio de herramientas similares: 4 días.
    - Elaboración de la documentación: 4 días.

A continuación se mostrarán los elementos en el diagrama de Gantt:

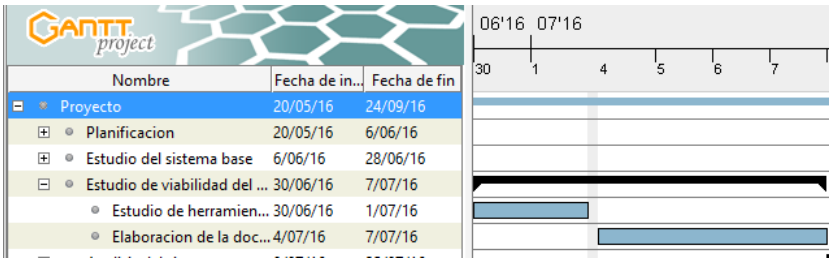

Ilustración 31. Diagrama de Gantt – estudio de viabilidad del sistema.

5.1.5. Análisis del sistema

El análisis del sistema es una de las fases más importantes del proyecto, ya que se establecen los requisitos y los objetivos del proyecto. Se compondrá de los siguientes elementos:

- Análisis del sistema: 18 días.
    - Análisis del sistema: 8 días.
    - Elaboración de los requisitos: 5 días.
    - Elaboración de la documentación: 5 días.

El diagrama de Gantt de esta fase es el siguiente:

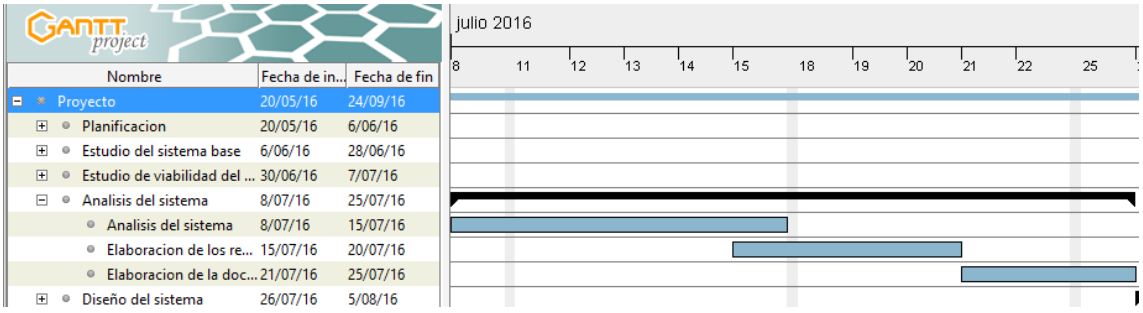

Ilustración 32. Diagrama de Gantt – análisis del sistema.

5.1.6. Diseño del sistema

Durante esta fase se diseñará y realizará la arquitectura del proyecto. Se compone de los siguientes elementos:

- Diseño del sistema: 11 días.
    - Reunión con el tutor: 2 días.
    - Diseño de la arquitectura: 5 días.
    - Elaboración de la documentación: 4 días.

Estos elementos se muestran en el siguiente diagrama de Gantt:

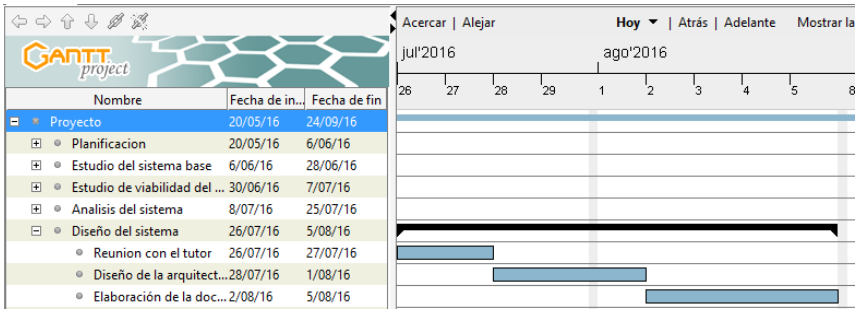

Ilustración 33. Diagrama de Gantt – diseño del sistema.

5.1.7. Implementación del sistema

Fase más extensa de todo el proyecto, es en la que se implementa todo lo diseñado en las fases anteriores, dando forma al sistema. Esta fase se compondrá de los siguientes elementos:

- Implementación del sistema: 30 días.
    - Reunión con el tutor: 2 días.
    - Implementación del sistema de comunicación: 4 días.
    - Implementación del cliente y el servidor: 20 días.
    - Elaboración de la documentación: 4 días.

El diagrama de Gantt de esta fase es el siguiente:

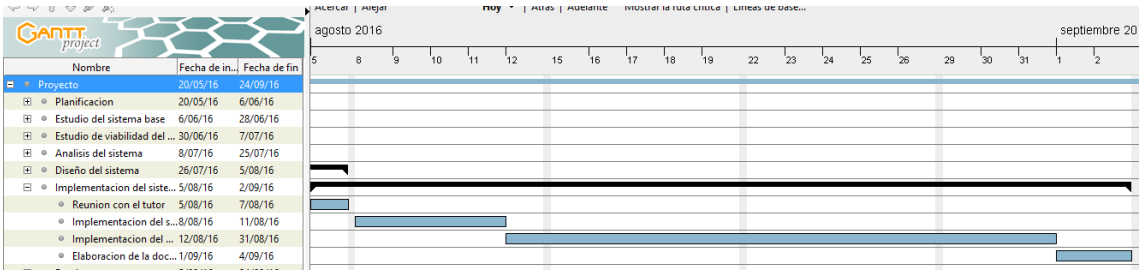

Ilustración 34. Diagrama de Gantt – implementación del sistema.

5.1.8. Pruebas

Fase final del proyecto, en él se estipulan las pruebas que determinaran el buen funcionamiento del sistema. Se compone de los siguientes elementos:

- Pruebas: 20 días.
    - Reunión con el tutor: 1 día.
    - Definición de las pruebas: 2 días.
    - Implementación de las pruebas: 7 días.
    - Análisis de los resultados obtenidos: 5 días.
    - Elaboración de la documentación: 5 días.

El diagrama de esta fase es el siguiente:

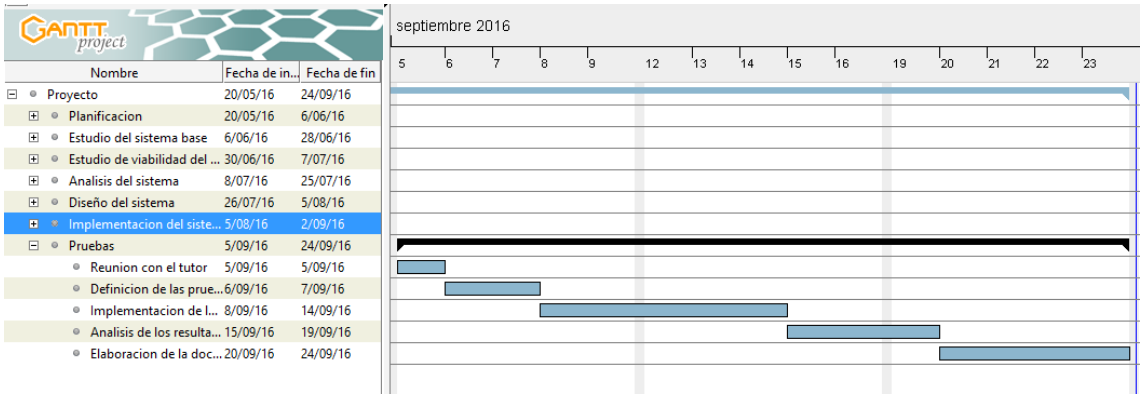

Ilustración 35. Diagrama de Gantt – pruebas.

5.2. COCOMO

El Modelo Constructivo de Costos (o COCOMO) es un modelo matemático utilizado para estimación de costos de software a partir de tres submodelos, ofreciendo cada uno un nivel de aproximación y detalle cada vez mayor, a medida que el proceso de desarrollo del software va avanzando (de nivel básico a intermedio o detallado).

Este modelo basa la estimación no en el coste por tiempo, si no en el coste por líneas de código, lo cual ofrece unos resultados no proporcionales a las tareas de gestión. La mayor parte de nuestro proyecto se basa en este tipo de tareas –aprendizaje de uso de librerías, elección de estas y análisis del sistema base.

Además, el tamaño de líneas programadas en la realización de este proyecto es relativamente bajo con lo usual en proyectos analizados con COCOMO; como nos avisa la propia aplicación COCOMO II mediante un aviso el cual nos indica que si el tamaño de líneas es menor de 2000 no es recomendable usar esta herramienta; por lo que no se puede estimar, con buenos resultados, el coste de nuestro proyecto mediante COCOMO.

6. Evaluación

En el siguiente apartado se realizará un análisis del funcionamiento del sistema mediante pruebas que se realizarán sobre él. Además, se especificará la plataforma usada para su realización.

6.1. Descripción de la plataforma

A la hora de la realización de las pruebas es de vital importancia especificar en qué plataforma se están realizando. Esto es debido a que puede haber valores, o comportamientos, que dependan de la arquitectura de la plataforma en la que se ejecute, tanto hardware como software, por lo que a continuación se especificará la plataforma en su totalidad.

6.1.1. Hardware

El hardware utilizado ha sido el siguiente:

- ***Toshiba Satellite***
  - Procesador Intel Core i3-3110.
  - Velocidad de procesador: 2.4 GHz.
  - Memoria: 8GB.
  - Disco Duro: 1TB.
  - Sistema Operativo: Linux Ubuntu.
  - Tarjeta Gráfica: Intel HD Graphics.

6.1.2. Software

El software necesario y utilizado a la hora de la realización de las pruebas es el siguiente:

- **Sistema operativo:** Linux Ubuntu.
- **Entornos**: Java y C.
- **Librerías:**
  - MPICH2 v1.4.1
  - PAPI v5.1.1
  - Libxml2 v2.9.1
  - JFreeChart
  - WEKA3

6.2. Diseño de pruebas

En este apartado se diseña las pruebas mínimas para comprobar el correcto funcionamiento del sistema. Para ello, seguiremos un patrón similar a los requisitos, definiendo las pruebas a realizar mediante la siguiente plantilla.

PR-XXXX	
Titulo	
Descripción	
Resultado esperado	
Resultado obtenido	
Dependencias	

Tabla 24. Patrón de prueba.

Los parámetros de estas pruebas son los siguientes:

- ***Código***: código único que representa la prueba. Ese código se compone por los caracteres PR (Prueba), junto a dos caracteres que indicará prueba de funcionamiento (PF) o prueba de rendimiento (PR), y un número que le identifica dentro de esos subgrupos.
- ***Título:*** Título que representa el objetivo de la prueba.
- ***Descripción:*** breve explicación del carácter de la prueba.
- ***Resultado esperado:*** resultado que debería dar la prueba.
- ***Resultado obtenido:*** resultado que ha devuelto la prueba.
- ***Dependencias:*** pruebas o requisitos de los que depende la prueba.

6.2.1. Especificación de pruebas

Como se ha especificado antes, se procederá a la especificación de las pruebas de la misma manera que los requisitos. Además, se clasificarán en dos grupos, dependiendo de su funcionalidad. Esto es así porque no únicamente se tiene que probar la funcionalidad básica, si no que el rendimiento y los resultados son los esperados, por lo que se tendrán en cuenta los siguientes casos:

- ***Pruebas de funcionamiento:*** estas pruebas básicas tendrán el objetivo de comprobar el correcto funcionamiento del sistema, en base a los requisitos de sistema especificados en el capítulo 3 – “Descripción de la arquitectura – Análisis de requisitos”.
- ***Pruebas de rendimiento***: estas pruebas tendrán el objetivo de esclarecer el rendimiento mejorado – o empeorado - mediante la comparación de nuestro sistema frente al sistema original; en base a los requisitos de sistema especificados en el capítulo 3. Para ello, se utilizará como aplicación a monitorizar el programa *gradient*, el cual realizará escrituras de bloques de tamaño configurable como evento de E/S.

Pruebas de funcionamiento

PR-PF01	
Titulo	Inicialización librería MPI.
Descripción	La librería debe inicializar la librería MPI original sin errores.
Resultado esperado	La inicialización no producirá errores.
Resultado obtenido	
Dependencias	RS-FU01, RS-FU06, RS-FU02, RS-FU03

Tabla 25. Prueba PR-PF01.

PR-PF02	
Titulo	Inicialización librería PAPI.
Descripción	La librería debe inicializar la librería PAPI sin errores.
Resultado esperado	La inicialización no producirá errores.
Resultado obtenido	
Dependencias	RS-FU01, RS-FU06, RS-FU02

Tabla 26. Prueba PR-PF02.

PR-PF03	
Titulo	Inicialización librería WEKA.
Descripción	La librería debe inicializar la librería WEKA sin errores.
Resultado esperado	La inicialización no producirá errores.
Resultado obtenido	
Dependencias	RS-FU01, RS-FU04, RS-FU14, RS-FU12, RS-FU13

Tabla 27. Prueba PR-PF03.

PR-PF04	
Titulo	Creación del hilo.
Descripción	Durante la inicialización de la librería se ha de crear el hilo correspondiente.
Resultado esperado	La creación de un hilo y su ejecución.
Resultado obtenido	
Dependencias	RS-FU01, RS-FU05

Tabla 28. Prueba PR-PF04.

PR-PF05	
Titulo	Creación del mutex sobre el búfer.
Descripción	Durante la inicialización de la librería se creará el mutex correspondiente.
Resultado esperado	La creación de un mutex.
Resultado obtenido	
Dependencias	RS-FU05

Tabla 29. Prueba PR-PF05.

PR-PF06	
Titulo	Creación del mutex sobre los flags.
Descripción	Durante la inicialización del servidor, se creará el mutex correspondiente.
Resultado esperado	La creación de un mutex.
Resultado obtenido	
Dependencias	RS-FU05

Tabla 30. Prueba PR-PF06.

PR-PF07	
Titulo	Inicialización aleatoria de la función hash.
Descripción	Durante la inicialización de la librería se debe inicializar la función hash de forma aleatoria.
Resultado esperado	Cada ejecución inicializa esta función y produce un resultado distinto.
Resultado obtenido	
Dependencias	RS-FU06

Tabla 31. Prueba PR-PF07.

PR-PF08	
Titulo	Inicialización del modelo de WEKA.
Descripción	Durante la ejecución del servidor se inicializará el modelo Random Tree de WEKA.
Resultado esperado	Cada hilo inicia un modelo de datos y no devuelve error.
Resultado obtenido	
Dependencias	RS-FU01, RS-FU04, RS-FU14

Tabla 32. Prueba PR-PF08.

PR-PF09	
Titulo	Crear el set de eventos de PAPI.
Descripción	Se creará el set de eventos en el que se indicarán los contadores a leer.
Resultado esperado	Se crea el set de eventos y no devuelve error.
Resultado obtenido	
Dependencias	RS-FU02, RS-FU03, RS-FU06

Tabla 33. Prueba PR-PF09.

PR-PF10	
Titulo	Añadir eventos al set de eventos de PAPI.
Descripción	Se añadirá el evento que indicará que contador leer.
Resultado esperado	Se añadirá sin ningún error.
Resultado obtenido	
Dependencias	RS-FU02, RS-FU03, RS-FU06

Tabla 34. Prueba PR-PF10.

PR-PF11	
Titulo	Inicialización de contadores PAPI.
Descripción	Los contadores se deben inicializar sin errores para poder leerlos.
Resultado esperado	Se inicializará sin ningún error.
Resultado obtenido	
Dependencias	RS-FU02, RS-FU03, RS-FU06

Tabla 35. Prueba PR-PF11.

PR-PF12	
Titulo	Lectura de contadores PAPI.
Descripción	Se obtendrá una lectura coherente de los contadores PAPI.
Resultado esperado	Se obtiene un número coherente.
Resultado obtenido	
Dependencias	RS-FU02, RS-FU03, RS-FU06, RS-FU04

Tabla 36. Prueba PR-PF12.

PR-PF13	
Titulo	Finalización de contadores PAPI.
Descripción	Se detendrá la cuenta de los contadores PAPI.
Resultado esperado	Se finaliza sin errores.
Resultado obtenido	
Dependencias	RS-FU01, RS-FU02, RS-FU03, RS-FU06

Tabla 37. Prueba PR-PF13.

PR-PF14	
Titulo	Finalización de la librería MPI.
Descripción	La librería debe finalizar la librería MPI original sin errores.
Resultado esperado	Se finaliza sin errores.
Resultado obtenido	
Dependencias	RS-FU01

Tabla 38. Prueba PR-PF14.

PR-PF15	
Titulo	Establecer conexión con el servidor.
Descripción	Se ha de comprobar si la conexión cliente-servidor se ha establecido correctamente, en el host y puerto correcto.
Resultado esperado	Se realiza la conexión correctamente.
Resultado obtenido	La conexión es realizada correctamente.
Dependencias	RS-FU07, RS-FU09, RS-FU11

Tabla 39. Prueba PR-PF15.

PR-PF16	
Titulo	Lectura de parámetros del fichero XML.
Descripción	Se ha de leer el contenido del fichero de configuración correctamente, comprobando sus parámetros.
Resultado esperado	Se leerá los parámetros correctamente.
Resultado obtenido	Los parámetros son leídos correctamente.
Dependencias	RS-FU11, RS-FU02, RS-FU07, RS-FU16, RS-FU09

Tabla 40. Prueba PR-PF16.

PR-PF17	
Titulo	Envío de mensajes al servidor.
Descripción	El cliente debe enviar los datos completos al servidor.
Resultado esperado	Se envía en su totalidad los datos al servidor.
Resultado obtenido	Todos los datos se envían, completos, al servidor.
Dependencias	RS-FU07, RS-FU09, RS-FU11

Tabla 41. Prueba PR-PF17.

PR-PF18	
Titulo	Recepción de mensajes del cliente.
Descripción	El servidor debe recibir los datos completos del cliente.
Resultado esperado	El servidor recibe, en su totalidad, los datos del cliente.
Resultado obtenido	El servidor recibe estos datos.
Dependencias	RS-FU07, RS-FU09, RS-FU11

Tabla 42. Prueba PR-PF18.

PR-PF19	
Titulo	Envío de mensajes al cliente.
Descripción	El servidor debe enviar el tiempo de espera al cliente sin fallos.
Resultado esperado	Se envía en su totalidad los tiempos de espera al cliente.
Resultado obtenido	El tiempo de espera se envía, completo, al cliente.
Dependencias	RS-FU07, RS-FU09, RS-FU11,RS-FU10

Tabla 43. Prueba PR-PF19.

PR-PF20	
Titulo	Recepción de mensajes del servidor.
Descripción	El cliente debe recibir el tiempo de espera del servidor sin errores.
Resultado esperado	El cliente recibe, en su totalidad, el tiempo de espera del servidor.
Resultado obtenido	El cliente recibe estos datos.
Dependencias	RS-FU07, RS-FU09, RS-FU11,RS-FU10

Tabla 44. Prueba PR-PF20.

PR-PF21	
Titulo	Comprobación del host.
Descripción	Se debe comprobar que el host es válido y conectable.
Resultado esperado	En caso de host no válido, se detiene la ejecución.
Resultado obtenido	Si el host es no válido, se detiene su ejecución.
Dependencias	RS-FU07, RS-FU09, RS-FU11

Tabla 45. Prueba PR-PF21.

PR-PF22	
Titulo	Recepción del puerto.
Descripción	Una vez establecida la conexión con el servidor se debe recibir un número de puerto para establecer la nueva conexión.
Resultado esperado	Se recibe el nuevo puerto.
Resultado obtenido	Se recibe el nuevo puerto y se conecta a él.
Dependencias	RS-FU07, RS-FU09, RS-FU11

Tabla 46. Prueba PR-PF22.

PR-PF23	
Titulo	Ejecución de una llamada MPI.
Descripción	Cada vez que se realiza una llamada a la biblioteca MPI original, se debe realizar sin ningún error.
Resultado esperado	Se realiza la llamada original sin ningún error.
Resultado obtenido	Se realiza la llamada original correctamente.
Dependencias	RS-FU01, RS-FU06, RS-FU02, RS-FU03

Tabla 47. Prueba PR-PF23.

PR-PF24	
Titulo	Comprobación de los contadores.
Descripción	Se debe comprobar que los contadores indicados existen y son correctos para la medida de eventos esperados en la aplicación.
Resultado esperado	En caso de contadores no válidos, se detiene la ejecución.
Resultado obtenido	Si el contador es no válido, se avisa al usuario y se detiene su ejecución.
Dependencias	RS-FU01, RS-FU06, RS-FU02, RS-FU03

Tabla 48. Prueba PR-PF24.

PR-PF25	
Titulo	Comprobación de la interfaz de usuario.
Descripción	Se comprueba el correcto funcionamiento de la interfaz del usuario, así como la correcta muestra de los datos y las gráficas.
Resultado esperado	Se muestran los datos y la gráfica correctamente.
Resultado obtenido	Los datos y la gráfica mostrada se corresponden con lo almacenado en la matriz.
Dependencias	RS-FU08, RS-FU15, RS-FU16

Tabla 49. Prueba PR-PF25.

PR-PF26	
Titulo	Procesamiento de cada hilo.
Descripción	Cada vez que se reciba una petición al servidor, esta se debe tratar con la creación de un nuevo hilo.
Resultado esperado	Cada hilo recibe un puerto distinto.
Resultado obtenido	Para cada hilo del cliente se crea un hilo en el servidor con un puerto distinto.
Dependencias	RS-FU05, RS-FU08, RS-FU09

Tabla 50. Prueba PR-PF26.

PR-PF27	
Titulo	Establecimiento de tiempo de espera.
Descripción	Cuando la librería recibe un tiempo de espera, se establecerá en el cliente.
Resultado esperado	Se almacena el tiempo de espera.
Resultado obtenido	Cuando el cliente recibe un tiempo de espera, se almacena en él.
Dependencias	RS-FU05, RS-FU04, RS-FU09, RS-FU10

Tabla 51. Prueba PR-PF27.

PR-PF28	
Titulo	Suspensión del proceso.
Descripción	El proceso cliente se mantendrá en suspensión cuando reciba un tiempo de espera.
Resultado esperado	El proceso se suspende si hay tiempo de espera.
Resultado obtenido	El proceso se suspende durante el tiempo recibido por el servidor. Luego se despierta.
Dependencias	RS-FU05, RS-FU04, RS-FU09, RS-FU10

Tabla 52. Prueba PR-PF28.

PR-PF29	
Titulo	Detección de E/S.
Descripción	El servidor debe ser capaz de detectar en los contadores eventos de E/S.
Resultado esperado	El servidor detectará en las subidas o bajadas de los contadores los eventos E/S.
Resultado obtenido	Cuando se produce un pico en los contadores, el servidor detecta un evento de E/S.
Dependencias	RS-FU02, RS-FU03, RS-FU04, RS-FU06

Tabla 53. Prueba PR-PF29.

PR-PF30	
Titulo	Predicción de E/S.
Descripción	El servidor debe ser capaz de predecir un gran número de eventos E/S.
Resultado esperado	El servidor predice los eventos E/S mediante el modelo de datos.
Resultado obtenido	Cuando el modelo de datos da predicción positiva, indica evento de E/S.
Dependencias	RS-FU02, RS-FU03, RS-FU04, RS-FU06, RS-FU13, RS-FU14

Tabla 54. Prueba PR-PF30.

PR-PF31	
Titulo	Actualización del modelo de predicción.
Descripción	El servidor deberá actualizar el modelo de predicción con los datos guardados.
Resultado esperado	En cada iteración del servidor, se ha de actualizar el modelo de datos.
Resultado obtenido	En cada iteración del servidor se ofrece un modelo de datos distinto.
Dependencias	RS-FU04, RS-FU13, RS-FU14, RS-FU13

Tabla 55. Prueba PR-PF31.

PR-PF32	
Titulo	Creación de fichero arff.
Descripción	El servidor deberá crear un fichero arff por cada hilo creado, en el que guardará los datos de los eventos.
Resultado esperado	Se creará el fichero sin errores.
Resultado obtenido	El fichero es creado sin ningún error.
Dependencias	RS-FU04, RS-FU13, RS-FU14, RS-FU12

Tabla 56. Prueba PR-PF32.

PR-PF33	
Titulo	Lectura de fichero arff.
Descripción	El servidor deberá leer los datos contenidos en el fichero arff sin errores.
Resultado esperado	Los datos del arff se leen correctamente.
Resultado obtenido	Los datos del arff leídos se corresponden a los contenidos en el fichero.
Dependencias	RS-FU04, RS-FU13, RS-FU14, RS-FU12

Tabla 57. Prueba PR-PF33.

PR-PF34	
Titulo	Guardado de datos.
Descripción	El servidor deberá guardar los datos de los contadores en el archivo arff sin ningún error.
Resultado esperado	El servidor guardará los datos leídos en el fichero arff.
Resultado obtenido	El contenido del fichero arff se corresponden a los contadores leídos por el servidor.
Dependencias	RS-FU04, RS-FU13, RS-FU14, RS-FU12, RS-FU03, RS-FU06

Tabla 58. Prueba PR-PF34.

PR-PF35	
Titulo	Establecimiento del estado del proceso.
Descripción	El servidor controlará el estado de cada cliente conectado.
Resultado esperado	El servidor tendrá información del estado de cada cliente.
Resultado obtenido	La información que tiene el servidor de cada cliente es independiente.
Dependencias	RS-FU03, RS-FU04, RS-FU06, RS-FU13, RS-FU14

Tabla 59. Prueba PR-PF35.

PR-PF36	
Titulo	Comprobación de los procesos en espera.
Descripción	El servidor llevará la cuenta de los procesos que están suspendidos o en proceso de suspenderse.
Resultado esperado	Cada vez que hay un proceso suspendido se cuantificará.
Resultado obtenido	El número de procesos suspendidos se corresponden con la cuenta.
Dependencias	RS-FU14, RS-FU10, RS-FU04

Tabla 60. Prueba PR-PF36.

PR-PF37	
Titulo	Restablecimiento de valores final.
Descripción	Al finalizar un hilo, el servidor reestablecerá los datos de este.
Resultado esperado	Al finalizar el hilo, los valores vuelven a 0.
Resultado obtenido	Al finalizar el hilo, los valores han vuelto a 0.
Dependencias	RS-FU01, RS-FU05

Tabla 61. Prueba PR-PF37.

Pruebas de rendimiento

PR-PR01	
Titulo	Prueba básica.
Descripción	Prueba con un único proceso.
Resultado esperado	Ejecución sin errores de un único proceso.
Resultado obtenido	Se ejecuta sin errores el proceso.

Tabla 62. Prueba PR-PR01.

PR-PR02	
Titulo	Prueba con colisiones bajas y tamaño de bloque de escritura pequeño.
Descripción	Se realiza la ejecución del servidor con el rango de 2-3 clientes, produciendo colisiones bajas; y el bloque de escritura de estos clientes es de tamaño bajo.
Resultado esperado	Ligera mejora del rendimiento respecto al original, debido a las pocas colisiones.
Resultado obtenido	El rendimiento es ligeramente peor respecto al original.

Tabla 63. Prueba PR-PR02.

PR-PR03	
Titulo	Prueba con colisiones bajas y tamaño de bloque de escritura grande.
Descripción	Se realiza la ejecución del servidor con el rango de 2-3 clientes, produciendo colisiones bajas; y el bloque de escritura de estos clientes es de tamaño grande.
Resultado esperado	Mejora del rendimiento, pero manteniéndose en valores bajos.
Resultado obtenido	

Tabla 64. Prueba PR-PR03.

PR-PR04	
Titulo	Prueba con colisiones media y tamaño de bloque de escritura pequeño.
Descripción	Se realiza la ejecución del servidor con el rango de 4-8 clientes, produciendo colisiones más frecuentemente; y el bloque de escritura de estos clientes es de tamaño pequeño.
Resultado esperado	Al aumentar ligeramente las colisiones, el rendimiento empeora, pero manteniéndose mejor que el original en el mismo caso.
Resultado obtenido	El rendimiento es ligeramente mejor a las colisiones pequeñas pero sigue siendo negativo.

Tabla 65. Prueba PR-PR04.

PR-PR05	
Titulo	Prueba con colisiones media y tamaño de bloque de escritura grande.
Descripción	Se realiza la ejecución del servidor con el rango de 4-8 clientes, produciendo colisiones más frecuentemente; y el bloque de escritura de estos clientes es de tamaño grande.
Resultado esperado	Al aumentar ligeramente las colisiones y el tamaño del bloque alto, el rendimiento empeora, pero manteniéndose mejor que el original en el mismo caso.
Resultado obtenido	El rendimiento es ligeramente mejor respecto al sistema original.

Tabla 66. Prueba PR-PR05.

PR-PR06	
Titulo	Prueba con colisiones altas y tamaño de bloque de escritura pequeño.
Descripción	Se realiza la ejecución del servidor con el rango de 10 clientes, produciendo colisiones muy frecuentes; y el bloque de escritura de estos clientes es de tamaño pequeño.
Resultado esperado	Al aumentar considerablemente las colisiones, el rendimiento empeora, pero manteniéndose mejor que el original en el mismo caso.
Resultado obtenido	El rendimiento es mucho mejor que respecto a las anteriores colisiones, pero sigue siendo negativo.

Tabla 67. Prueba PR-PR06.

PR-PR07	
Titulo	Prueba con colisiones altas y tamaño de bloque de escritura pequeño.
Descripción	Se realiza la ejecución del servidor con el rango de 10 clientes, produciendo colisiones muy frecuentes; y el bloque de escritura de estos clientes es de tamaño grande.
Resultado esperado	Al aumentar considerablemente las colisiones, el rendimiento empeora, pero manteniéndose mejor que el original en el mismo caso.
Resultado obtenido	

Tabla 68. Prueba PR-PR07.

6.2.2. Matriz de trazabilidad

Tras la definición de las pruebas, en este apartado se muestra la matriz de trazabilidad que relaciona los requisitos de capacidad con las pruebas del sistema.

	RS-FU01	RS-FU02	RS-FU03	RS-FU04	RS-FU05	RS-FU06	RS-FU07	RS-FU08	RS-FU09	RS-FU10	RS-FU11	RS-FU12	RS-FU13	RS-FU14	RS-FU15	RS-FU16
PR-PF01	X	X	X			X										
PR-PF02	X	X				X										
PR-PF03	X			X								X	X	X		
PR-PF04	X				X											
PR-PF05					X											
PR-PF06					X											
PR-PF07						X										
PR-PF08	X			X										X		
PR-PF09		X	X			X										
PR-PF10		X	X			X										
PR-PF11		X	X			X										
PR-PF12		X	X	X		X										
PR-PF13	X	X	X			X										
PR-PF14	X															
PR-PF15							X		X		X					
PR-PF16		X					X		X		X					X
PR-PF17							X		X		X					
PR-PF18							X		X		X					
PR-PF19							X		X	X	X					
PR-PF20							X		X	X	X					
PR-PF21							X		X		X					
PR-PF22							X		X		X					
PR-PF23	X	X	X			X										
PR-PF24	X	X	X			X										
PR-PF25								X							X	X
PR-PF26					X			X	X							
PR-PF27				X	X				X	X						
PR-PF28				X	X				X	X						
PR-PF29		X	X	X		X										
PR-PF30		X	X	X		X							X	X		
PR-PF31				X								X	X	X		
PR-PF32				X								X	X	X		
PR-PF33				X								X	X	X		
PR-PF34			X	X		X						X	X	X		
PR-PF35			X	X		X							X	X		
PR-PF36					X					X			X			
PR-PF37	X				X											

Tabla 69. Matriz de trazabilidad del sistema.

6.3. Análisis de los resultados

En este apartado explicaremos las pruebas realizadas, así como los programas utilizados para ellas y la configuración. Estas pruebas se realizarán en el ordenador portátil *Toshiba Satellite*, ya que es que contiene el Sistema Operativo Linux; en el cual se ejecutará en un proceso el servidor, y en varios procesos – dependiendo de las pruebas – la aplicación a monitorizar.

Esta aplicación es una variante del método del gradiente conjugado (*Rapin, 2011*), un método iterativo usado para resolver grandes sistemas de ecuaciones a partir de una matriz de entrada; en el caso de las pruebas la matriz elegida es *nd7k.* Esta aplicación permite también indicar el número de iteraciones que queremos realizar para completar la función para la que está programada la aplicación: minimizar la función $f(x) = \frac{1}{2}x^t Ax - b^T x$.

El gradiente realiza, además, las operaciones de cargar la matriz, iniciar y finalizar MPI; por lo que se ejecuta de forma distribuida en varios procesos. Respecto a las pruebas de nuestro sistema nos interesa el gradiente conjugado está implementado en la función *cq*, definida en pseudocódigo:

- *For it = 0, 1, …, itmax*
  1. *If it = 0*
     - *r = b – Ax*
     - *s = A * x*
     - *r = -1 * s + b*
     - *s = r*
     - *z = zeros*
     - *r * r*
  2. *If it % intervalo = 0*
     - *For j = 0,1,…,count*
       - *For k= 0, 1, …, BLOCKSIZE*
         - *Escribir en disco valores de s*
  3. *Else*
     - *z = a * s*
     - *α = (r*r)/(s*z)*
     - *x = α * s + x*
     - *r' = -α * z + r*
     - *ρ = (r'*r')/(r*r)*
     - *s = ρ * s + r'*
     - *r = r'*

Ilustración 36. Pseudocódigo gradient.

El algoritmo original fue modificado para que realizara una operación de E/S de escritura en la que escribe los valores del resultado. Esta lógica nueva emula una operación de *checkpointing* de la aplicación.

Cada ejecución de esta aplicación se realizará con los siguientes parámetros, los cuales asignaremos a la hora de ejecutar la aplicación:

- **Tamaño del archivo de datos** = 18000.
- **Número de iteraciones** = 2000.
- **Precisión** = 0.002.

La configuración establecida para la realización de las pruebas son las siguientes:

- **Iteraciones *(itmax)*** = número de iteraciones entre escrituras. Se mantendrá constante en 200.
- **Tamaño de bloque *(BLOCKSIZE)*** = factor de escala sobre la estructura originalmente almacenada. En las pruebas tendremos tamaño pequeño (*1.000*) y grande (*10.000*). Estos valores representan el número de veces que se repite la escritura del valor contenido en cada posición de s (*doubles*).
- **Numero de hilos**: número de hilos de control por proceso que se ejecutan. Se mantendrá estable en las pruebas con un único hilo por proceso.
- **Tiempo de espera**: la mayor parte de las pruebas usará el tiempo entre ejecuciones de eventos de e/s como tiempo de espera. Si los resultados no son los esperados, se probará con el tiempo en el que tarda en ejecutarse estos eventos.
- **Numero de aplicaciones**: número de aplicaciones que lanzamos. Tendrá los siguientes valores:
    - ***Bajo***: 3 procesos.
    - ***Media***: 4 procesos.
    - ***Alta***: 10 procesos.

Con estos parámetros, procederemos a recoger tanto el tiempo de ejecución de la aplicación con el nuevo sistema, como el tiempo de ejecución con el sistema anterior, y la diferencia de rendimiento entre los dos sistemas.

Para ello dividiremos las pruebas en dos grupos, dependiendo del tamaño de bloque que hemos utilizado, para así ver en rendimiento en caso de carga baja o carga grande.

6.3.1. Pruebas de carga baja.

El objetivo de estas pruebas es comprobar si el sistema definido durante todo el proyecto mejora, o empeora, el rendimiento en aplicaciones que acceden al mismo recurso a la vez. En este primer caso, el uso del recurso compartido llevará un tiempo corto, debido al pequeño tamaño de las escrituras a disco.

Los datos que mostraremos en ellas serán una media de las cinco ejecuciones de las pruebas. Esto es debido a que el tiempo de ejecución no depende únicamente del sistema, si no que puede verse alterado por ejecuciones en segundo plano de la computadora, o por tareas del sistema. Se mostrará la media, la varianza y la desviación típica de estos datos.

La configuración, como se ha estipulado en el apartado anterior, será la siguiente:

Iteraciones	Tamaño bloque	Hilos
200	1000	1

Tabla 70. Parámetros de configuración de pruebas de carga baja.

A continuación se mostrarán los resultados obtenidos mediante esta configuración, clasificados por número de procesos.

Bajo número de procesos.

Estos resultados son la media de los resultados obtenidos por 3 procesos ejecutándose a la vez.

Sistema	Original			Actual		
Procesos	Media	Desviación estándar	Varianza	Media	Desviación estándar	Varianza
Proceso 1	1651.2	39.0346	1523.7	1732.8	19.8	392.2
Proceso 2	1759.4	26.29258	691.3	1799.6	24.65	607.8
Proceso 3	1760.6	23.29807	542.8	1812.8	21.833	476.7
Media total	1723,73	29,54	919,27	1781,73	22,09	492,23

Tabla 71. Resultados de pruebas de carga baja – número reducido de procesos.

El rendimiento total del sistema es:

Media original	Media actual	Rendimiento
1723,73	1781,73	-3,36%

Tabla 72. Rendimiento de pruebas de carga baja – número reducido de procesos.

Como podemos observar, en una primera prueba de poca carga con pocos datos no es recomendable el uso del sistema, ya que nos ofrece un rendimiento peor que si ejecutáramos el sistema original, es decir, con colisiones a la hora de acceder a un recurso, ya que puede ser que el tiempo que el servidor manda esperar al cliente sea mayor que lo que tardaría en completar la tarea estando compartiendo recurso.

Número medio de procesos.

Estos resultados son la media de los resultados obtenidos por 4 procesos ejecutándose a la vez.

Sistema	Original			Actual		
Procesos	Media	Desviación estándar	Varianza	Media	Desviación estándar	Varianza
Proceso 1	1753	18,38	338,24	1927	19,8	392
Proceso 2	1850	24,75	612,78	1874	12,73	162
Proceso 3	1895	16,26	264,56	1908	11,31	128
Proceso 4	1901	21,21	450,35	1935	21,21	450
Media total	1849,75	20,15	416,48	1911,00	16,26	283,00

Tabla 73. Resultados de pruebas de carga baja – número medio de procesos.

El rendimiento total del sistema es:

Media original	Media actual	Rendimiento
1849,75	1911,00	-3,31%

Tabla 74. Rendimiento de pruebas de carga baja – número medio de procesos.

Como se puede observar, el rendimiento ha mejorado ligeramente respecto al anterior caso, aunque sigue siendo negativo y, por lo tanto, sigue sin ser recomendable su uso. Que haya disminuido puede significar que nuestro sistema es más beneficioso cuanta más colisiones, o tiempo de compartición, tenga.

Alto número de procesos.

Estos resultados son la media de los resultados obtenidos por 10 procesos ejecutándose a la vez.

Sistema	Original			Actual		
Procesos	Media	Desviación estándar	Varianza	Media	Desviación estándar	Varianza
Proceso 1	3058	6,02	36,3	2967	75	5633
Proceso 2	3060,6	4,16	19,3	3038	94,5	8946
Proceso 3	3074,67	12,85	165,33	3169	107,39	11533
Proceso 4	3107,3	16,07	258,33	3202	90,87	8258
Proceso 5	3173,6	57,7	3330,33	3252	63,7	4056
Proceso 6	3249,3	28,99	840,33	3278	38,5	1486,3
Proceso 7	3266,67	10,5	110,33	3321	16,56	274,3
Proceso 8	3269,67	9,61	92,33	3365,7	67,28	4526,3
Proceso 9	3275	12,29	151	3379,3	65,38	4274,3
Proceso 10	3425	159,21	25348	3411,67	41,4	1714,3
Media total	3195,98	31,74	3035,16	3238,37	66,06	5070,15

Tabla 75. Resultados de pruebas de carga baja – número alto de procesos.

El rendimiento total del sistema es:

Media original	Media actual	Rendimiento
3195,98	3238,37	-1,33%

Tabla 76. Rendimiento de pruebas de carga baja – número alto de procesos.

Otra vez nos encontramos con un rendimiento mejorable respecto al caso anterior, lo cual confirma que el beneficio obtenido por nuestra herramienta es mayor dependiendo de la carga de compartición que tenga el sistema.

6.3.2. Pruebas de carga alta.

En este segundo caso, el uso del recurso compartido llevará un tiempo ya considerable, debido al gran tamaño de los bloques de escrituras a disco.

Por lo que hemos podido ir observando en los casos anteriores, el beneficio es mayor aumentando el número de procesos que acceden al recurso, pero aun así sigue siendo un rendimiento negativo. Con estas pruebas se podrá observar el comportamiento del sistema y los beneficios dependiendo tanto de los procesos como de una carga de escritura mucho mayor.

La configuración, como se ha estipulado en el apartado anterior, será la siguiente:

Iteraciones	Tamaño bloque	Hilos
200	10000	1

Tabla 77. Parámetros de configuración de pruebas de carga alta.

A continuación se mostrarán los resultados obtenidos mediante esta configuración, clasificados por número de procesos.

Bajo número de procesos.

Estos resultados son la media de los resultados obtenidos por 3 procesos ejecutándose a la vez.

Sistema	Original			Actual		
Procesos	Media	Desviación estándar	Varianza	Media	Desviación estándar	Varianza
Proceso 1	7964	13669	116,91	7732,66	5574,33	74,66
Proceso 2	8120,33	5032,33	70,94	7977,67	56,33	7,51
Proceso 3	8118,66	5497,33	74,144	8004,33	1066,33	32,65
Media total	8067,66	8066,22	87,33	7904,89	2232,33	38,27

Tabla 78. Resultados de pruebas de carga alta – número bajo de procesos.

El rendimiento total del sistema es:

Media original	Media actual	Rendimiento
8067,66	7904,89	2,02%

Tabla 79. Rendimiento de pruebas de carga alta – número bajo de procesos.

Como podemos observar, nos encontramos frente un rendimiento positivo de nuestra herramienta frente a la original, aunque se trata de un rendimiento muy bajo. Esto indica que, a mayor carga de datos, mejor comportamiento de la herramienta y más beneficio ofrece; aunque es muy pronto para afirmar si se da este caso en el resto de pruebas.

Número medio de procesos.

Estos resultados son la media de los resultados obtenidos por 4 procesos ejecutándose a la vez.

Sistema	Original			Actual		
Procesos	Media	Desviación estándar	Varianza	Media	Desviación estándar	Varianza
Proceso 1	8938,3	150,79	22737,3	8832,66	52,53	2760,33
Proceso 2	9157	29,72	883	8968	53,03	2812
Proceso 3	9170	19,52	381	9164,66	75,74	5736,3
Proceso 4	9175,67	16,04	257,33	9217,67	71,45	5105,33
Media total	9110,24	54,02	6064,66	9045,75	63,19	4103,49

Tabla 80. Resultados de pruebas de carga alta – número medio de procesos.

El rendimiento total del sistema es:

Media original	Media actual	Rendimiento
9110,24	9045,75	0,71%

Tabla 81. Rendimiento de pruebas de carga alta – número medio de procesos.

El rendimiento es positivo, lo cual vuelve a demostrar que con una carga global del sistema media – alta mantenemos beneficios respecto al rendimiento, en este caso, de un 0,71 % (alrededor de 1 minuto menos de ejecución global); siendo este más bajo que el caso anterior. Mediante la última prueba comprobaremos si la herramienta es útil con grandes procesos.

Alto número de procesos.

Estos resultados son la media de los resultados obtenidos por 10 procesos ejecutándose a la vez.

Sistema	Original			Actual		
Procesos	Media	Desviación estándar	Varianza	Media	Desviación estándar	Varianza
Proceso 1	23826,67	7952,33	89,17	23967	167,58	28086,33
Proceso 2	24095,66	16202,33	127,29	24079	196,41	38576,33
Proceso 3	24296,66	1774,33	42,12	24128,33	232,8	54194,33
Proceso 4	24360,33	5784,33	76,05	24259,66	184,23	33941,33
Proceso 5	24390,33	10257,33	101,29	24308	176,12	31021
Proceso 6	24397,67	9402,33	96,96	24364,33	129,81	16852,3
Proceso 7	24403	10507	102,5	24414,66	165,48	22737,3
Proceso 8	24448,66	9777,33	98,88	24448,66	173,36	30054,33
Proceso 9	24455,6	8953	94,62	24486,33	150,43	22629,33
Proceso 10	24596,33	39306,33	198,26	24543,33	173,78	30200,33
Media total	24327,09	11991,66	102,71	24299,93	175,00	30829,29

Tabla 82. Resultados de pruebas de carga alta – número alto de procesos.

El rendimiento total del sistema es:

Media original	Media actual	Rendimiento
24327,09	24299,93	0,11%

Tabla 83. Rendimiento de pruebas de carga alta – número alto de procesos.

El rendimiento ha vuelto a bajar aunque se mantiene en positivo, en el rango de un único 0,11% (menos de 30 segundos respecto al original). A continuación expondremos nuestras conclusiones respecto a las pruebas.

6.3.3. Conclusión

Tras estas pruebas, hemos obtenido los siguientes resultados, mostrados en la gráfica mostrada a continuación para poder observar su evolución a lo largo de las ejecuciones.

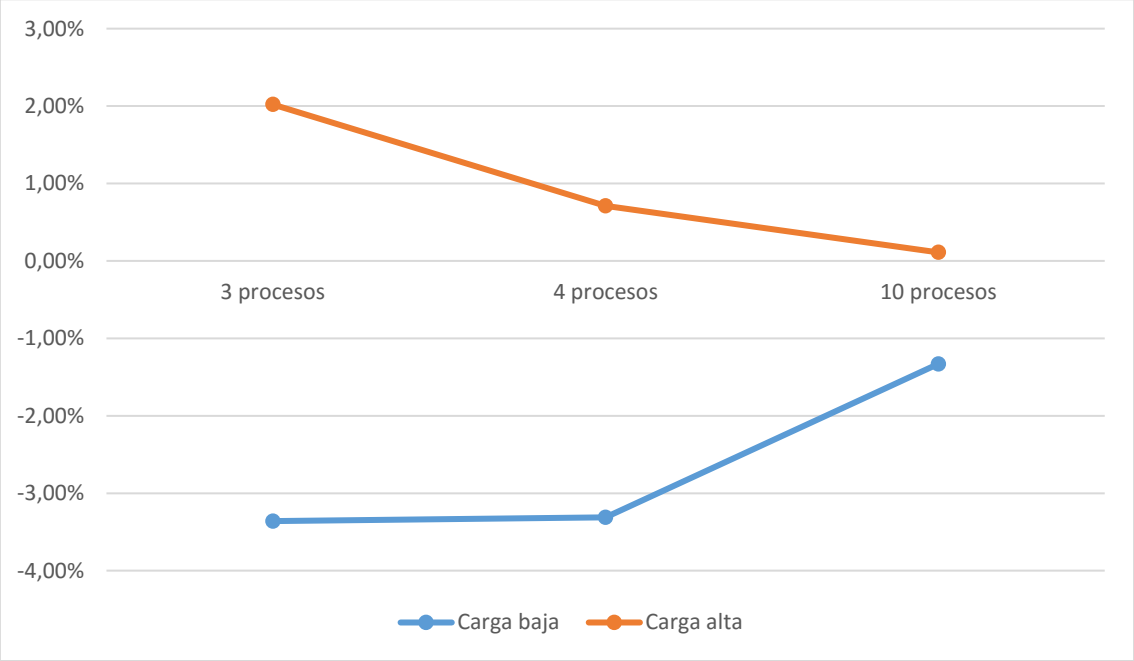

Ilustración 37. Gráfica de resultados.

Como podemos observar, existe una antítesis dependiendo de la carga que se le asigne a la E/S del hilo. Cuando tenemos aplicaciones que realizan cargas bajas, del rango de 1.000 como tamaño de bloque, obtenemos un beneficio de rendimiento mayor cuando más procesos ejecutemos al momento.

En cambio, cuando tenemos aplicaciones que realizan cargas elevadas, del rango de 10.000 como tamaño de bloque, podemos observar que el beneficio en rendimiento se va degradando a medida que añadimos procesos en paralelo.

El motivo de estos resultados puede residir en el tiempo de espera de los procesos enviado por el servidor. Este fue implementado de tal modo que se estableció ese tiempo en base al tiempo entre los eventos E/S, siendo el mismo tiempo en todas las implementaciones, ya que el número de iteraciones en todas las ejecuciones. Esta implementación se ha comprobado que no ofrece el beneficio esperado al sistema por los siguientes casos:

- Carga baja: el tiempo entre eventos de e/s es mayor al tiempo que tarda en ejecutarse estos eventos; lo que acarrea un tiempo de espera añadido al final cada ejecución a la hora de predecir. Esto puede verse reflejado en el hecho que la ejecución de 3 procesos tenga un rendimiento peor que 10 procesos, ya que el tiempo de ejecución de 3 procesos coincidiendo en paralelo acarrea menos tiempo que los 10 procesos, por lo tanto el extra añadido por esta espera es mayor en el caso de pocos procesos.
- Carga alta: en este caso ocurre justo todo lo contrario. El tiempo entre eventos e/s es menor que la duración de la ejecución de estos eventos, viéndose reflejado en que tenga una gráfica completamente opuesta al caso anterior. Esto acarrea un rendimiento mucho mejor en el caso de 3 procesos ya que la proporción de tiempo abarcada por el tiempo de suspensión respecto al tiempo de ejecución total es mucho mayor que en el caso de 10 procesos; ya que este último tarda más tiempo en ejecutarse y, como hemos dicho antes, el tiempo de espera es el mismo en ambos casos (200 iteraciones).

Visto esto, se pensó que el caso idóneo para solucionarlo sería mediante el cambio de este tiempo de espera o suspensión. Este tiempo, si se sustituye por el tiempo que tarda en ejecutarse los eventos e/s, haría que el sistema obtuviera una mejora en el rendimiento superior. Se procedió a implementar un modo en el que se tomara el tiempo de ejecución de los eventos, guardando siempre el menor de los tiempos, y usarlo como tiempo de espera. Sin embargo, los resultados obtenidos tampoco son favorables, como se puede observar en el siguiente resumen.

Procesos / Carga	Media tiempo entre e/s	Media tiempo ejecución e/s	Rendimiento
2 procesos/carga baja	1442,8	1446,9	-0,28%
3 procesos/carga baja	1781,73	1901	-6,69%

Tabla 84. Comparativa rendimiento entre tiempos de espera.

Esto es debido a que, aunque sea el tiempo de ejecución el que se utilice en las pruebas, este tiempo de ejecución se mantiene constante tanto si el cuello de botella se produce al inicio o al final de la ejecución del proceso que está utilizando el recurso, obteniendo una pérdida de tiempo, que acarrea una degradación del rendimiento mayor si se realiza la espera al final de la ejecución de este proceso.

En conclusión, las pruebas realizadas con los dos casos no han resultado ser las que se esperaban para este sistema. En el primer caso, si han ofrecido ligeros beneficios en caso de bloques de datos grandes; mientras que en el segundo la penalización ha sido global**.** Como posible solución a estos problemas respecto al tiempo de espera, se propone un tiempo dinámico, que no solo tenga en cuenta el tiempo de ejecución de cada proceso, sino además el porcentaje o porción de ese tiempo que ya ha sido ejecutado cuando se procese a pedir el acceso a ese recurso, minimizando al máximo el tiempo perdido al quedar libre el recurso.

7. Presupuesto

En este apartado se mostrará el presupuesto estimado del proyecto. Esto es crucial para el proyecto, ya que uno de los principales factores para la elección de un proyecto en el ámbito empresarial es el coste al que ascenderá su realización.

Este coste será detallado por categorías, dependiendo del objetivo del coste, para facilitar su análisis. Además, se han tenido en cuenta los impuestos en los mismos.

7.1. Mano de obra

Para realizar el presupuesto estimado de la mano de obra tendremos en cuenta las estimaciones de planificación temporal del capítulo 4. Para ello, estipularemos que cada día representado son **3 horas**.

Además, en el proyecto han participado dos personas:

- ***David Expósito Singh*** – tutor de proyecto.
- ***Alejandro García-Cantarero Alañón*** – alumno encargado de diseñar el proyecto.

Las horas estipuladas para cada uno depende de los días estipulados en las estimaciones mencionadas anteriormente, por lo que:

- ***David Expósito Singh:*** 37h.
- ***Alejandro García-Cantarero Alañón***: 381h.
- ***Total***: 418 horas.

Los costes de personal se estipularán en la siguiente tabla:

Categoría	Sueldo/hora*	Sueldo total	IRPF (21%)	SS (30%)	Coste/hora	Coste total
Tutor de Proyecto	32,5€	1202,50 €	252,52€	360,75€	49,07€	1815,77€
Alumno	18,75€	7143,75 €	1500,18€	2143,12€	21,70€	10787,05€
Total						12602,82

Tabla 85. Presupuesto – mano de obra.

Para el sueldo/hora se han seguido unas estimaciones de sueldo de Jefe de Proyecto (5200€/mes) y de Programador (3000 €/mes).

7.2. Hardware

En este apartado se especificarán los costes imputados al hardware utilizado en la realización del proyecto. En este caso se ha utilizado un ordenador portátil y un ordenador de sobremesa, junto a sus periféricos. Para el cálculo del coste imputable al proyecto, se tendrá en cuenta una vida útil de estos elementos de **3 años**, y una duración del proyecto de, aproximadamente, **4 meses**.

A continuación se mostrarán los costes imputables en la siguiente tabla.

Elemento	Coste total	Coste imputable
Ordenador de sobremesa	750,00€	83,33€
Ordenador portátil	500,00€	55,55€
Total		138,88€

Tabla 86. Presupuesto – hardware.

7.3. Software

En el siguiente apartado se especificarán los costes imputados a las herramientas software utilizadas en la realización del proyecto. Al trabajar con Linux, la gran parte de estas herramientas son de software libres, por lo que no tendrá ningún coste. Al software se le imputará una amortización de **3 años**, al igual que al hardware.

A continuación se mostrarán los costes imputables en la siguiente tabla.

Software	Precio	Coste imputable
Microsoft Windows 10*	121,50€	13,44€
Microsoft Office 365 *	79,00€	8,77€
Ubuntu	0,00€	0,00€
Eclipse	0,00€	0,00€
Notepad ++	0,00€	0,00€
Sublime Text 3	62,23€	6,91€
Total		29,12€

Tabla 87. Presupuesto – software.

Precios con 10% descuento para estudiantes en web oficial Microsoft.

7.4. Material fungible

En este apartado se estipularan los costes de material fungible que se atribuyen al proyecto, los cuales se pueden ver en la siguiente tabla.

Material	Coste
Bolígrafos	10€
Tinta	45€
Papel	4€
Cuadernos	4€
Otros gastos	100€
Total	163€

Tabla 88. Presupuesto – material fungible.

7.5. Resumen

A continuación se mostrará una tabla con un resumen de los gastos por sección, y el coste total estipulado al proyecto, además del presupuesto total, teniendo en cuenta un **beneficio del 10%,** un **riesgo del 5%**, y el **IVA actual (21%).**

Descripción	Coste
Mano de obra	12.602,82 €
Hardware	138,88 €
Software	29,12 €
Material fungible	163 €
Subtotal	12.933,82 €
Riesgo (5%)	646,69 €
Beneficio (10%)	1.293,38 €
Total sin IVA	14.873,89 €
IVA (21%)	3.123,52 €
Total	17.997,41 €

Tabla 89. Presupuesto – resumen total.

Por tanto, el presupuesto total para la realización de este proyecto se estipula en ***Diecisiete Mil Novecientos Noventa y Siete Euros con Cuarenta y Un Céntimos (IVA incluido).***

8. Conclusiones y trabajos futuros

En este apartado se expondrán las conclusiones obtenidas de la realización del proyecto, así como un pequeño resumen de su realización y describiremos los posibles trabajos futuros que se podrían realizar a partir de él.

8.1. Conclusiones

Este proyecto ha tenido como objetivo ampliar la herramienta de monitorización de aplicaciones para evitar la pérdida de rendimiento por compartición de recursos, en este caso, operaciones de E/S; y mejorar el rendimiento de las aplicaciones a partir de la predicción de este tipo de eventos y generar una planificación de las aplicaciones a partir de esta predicción.

El primer paso que se dio fue realizar un estudio y comprensión de la herramienta base, así como su código; además de realizar una investigación sobre herramientas similares en el mercado, y un estudio de las funcionalidades que ofrecen estas. A partir de este paso se estipularon los objetivos específicos del proyecto y, de ellos, se analizaron las librerías y los requisitos mínimos del sistema.

El paso a continuación era escoger en qué zona del código original implementar nuestra nueva funcionalidad. Primero se debía encontrar el método en el que se realizaba la comunicación cliente-servidor y, tras estudiar el protocolo MPI, se implementó una vía de comunicación bidireccional, mejorando la funcionalidad del sistema de comunicaciones y permitiendo que el servidor se pudiera comunicar con el cliente; cumpliendo así el objetivo de mejora del sistema de comunicaciones.

El siguiente objetivo que se debía cumplir era la detección de manera efectiva los eventos de E/S. Primero se analizó el comportamiento de las instrucciones de E/S y que componentes de la máquina se utilizaban a la hora de ejecutarlos. Se llegó a la conclusión que, cuando se realizaba una instrucción E/S no se realizaban operaciones FLOPS para el *benchmark* considerado. Finalmente había que implementar una lectura de esas instrucciones en nuestra herramienta. Esto fue posible gracias a la herramienta de monitorización original, que nos permitió comprobar que, un contador de operaciones FLOPS incluido en la librería PAPI presentaba un patrón de caída en el número de eventos cuando se ejecutaban estas instrucciones de E/S; cumpliéndose así el objetivo establecido.

A continuación, se debía estudiar el método estadístico para poder predecir cuándo uno de esos eventos iba a ocurrir. Para ello se estudiaron muchas tecnologías, llegando a la conclusión que la más idónea era el *Machine Learning,* a partir de la librería *WEKA,* la cual iba a ser utilizada para realizar esta predicción. Pero la librería WEKA ofrecía muchos modelos de predicción de datos, por lo que se tuvo que realizar una experimentación a partir de los datos recogidos por la herramienta original, para observar cuál era el modelo de datos más eficaz y preciso para la predecir, en nuestro caso, la realización de fases de E/S. La solución que se obtuvo fue en modelo *Random Tree*, ya que ofrecía menos error absoluto a la hora de predecir un evento. Gracias a esto se cumple el objetivo respecto a la elección del algoritmo de Machine Learning para predecir eventos.

Pero este sistema de predicción necesita de un conjunto de datos para poder funcionar correctamente. Para ello, la librería WEKA dispone de un formato de archivo, arff, para el almacenaje de este tipo de datos; y un catálogo de funciones para su creación, lectura, escritura e interacción con ficheros de ese tipo de formato. Gracias a esta gran funcionalidad de la librería WEKA es posible el cumplimiento del objetivo sobre la recogida de datos para el uso de Machine Learning.

Una vez predichos estos eventos, fue necesaria la elección de un tratamiento para intentar mejorar ese rendimiento perdido en el caso de cuello de botella y compartición de recursos. Se escogió el modo más sencillo: si varios procesos acceden a la vez a un recurso (en nuestro caso E/S), y a continuación hay un gran periodo de tiempo sin realizar ningún uso de ese recurso, por lo que se puede planificar los procesos que quieran acceder al recurso ocupado a esos periodos de tiempo en los que está desocupado. Eso fue posible mediante la implementación de una espera que mantiene el proceso en suspensión hasta que pasa un tiempo establecido, asegurando el objetivo que trata la reducción del tiempo de compartición de CPU.

Por último, todos estos datos recogidos de los contadores se muestran al usuario gracias a la funcionalidad de la interfaz de la herramienta base, por lo que el objetivo de muestra de la información a tiempo real se cumple desde un primer momento.

El conjunto de los cumplimientos de todos los objetivos específicos asegura el también cumplimiento del objetivo principal: crear una herramienta que prevé eventos de E/S y los trata para mejorar su rendimiento; aunque los resultados finales, como hemos comentado en el apartado correspondiente, no son los esperados debido a la errónea decisión del tiempo de espera.

Cabe destacar la visión a largo plazo de mejora de esta herramienta. Es una herramienta con un futuro muy amplio, tanto de funcionalidades como de mejorar el rendimiento de la herramienta y de las aplicaciones que monitoriza, como se tratará en el siguiente apartado. Para finalizar, se comentarán las experiencias adquiridas con la realización de este proyecto, además de las opiniones personales sobre la temática y la herramienta:

- Me ha permitido tener una conciencia superior acerca de la importancia del rendimiento para la creación e implementación de herramientas, tanto a la hora de escoger herramienta, como de ofrecerlas; ya que es un factor que se tiene muy en cuenta hoy en día.
- También me ha permitido observar lo frágil que es este rendimiento, ya que muchos factores, tanto de código, como de máquina o de conexión afectan a éste.
- Un proyecto de esta envergadura, además, me ha ofrecido un reto que me ha hecho evolucionar como estudiante, ya que es una de las primeras ocasiones que se da la oportunidad a un estudiante, en su carrera, de la creación de un proyecto tan importante.
- También he aprendido a asumir errores y contratiempos, adquiriendo mayor madurez y organización a la hora de resolver este tipo de problemas.
- He adquirido, por último, un mayor conocimiento de todas las ramas de la Ingeniería Informática, así como de tecnologías y librerías muy interesantes e importantes para proyectos futuros.

8.2. Trabajos futuros

Este proyecto está enfocado únicamente hacia un único problema de rendimiento: la compartición de recursos limitados; y dentro de esta, solo en instrucciones de E/S. En trabajos futuros se podría ampliar la funcionalidad de la herramienta para conseguir un mayor rango de problemas de rendimiento, mejorándolo y teniendo unos resultados muchos más favorables.

Otro trabajo futuro puede basarse en la librería WEKA. El ámbito de la tecnología Machine Learning está en constante crecimiento y cada breve espacio de tiempo nos encontramos con una nueva tecnología, o una versión mucho más mejorada. Probablemente, en cuestión de meses o años, aparece una tecnología de predicción de datos mucho más eficiente que la usada en este proyecto, y se pueden obtener con ella una predicción más eficaz, sin necesidad de tanto retardo como tiene este proyecto, mejorando así también la eficiencia del programa.

También se puede mejorar la lectura y detección de los eventos. En este proyecto se utiliza el contador que lee los FLOPS que se utilizan en la ejecución de la aplicación, pero ya existe una librería, extensión de la librería PAPI, llamada *appio*, la cual contiene herramientas de lectura de contadores de E/S muy interesantes. Esta librería podría ofrecer resultados más claros, ofreciendo también la posibilidad de simplificar el código, tanto de servidor como de la librería MPI modificada; mejorando el tiempo de ejecución de ambos programas y mejorando su rendimiento.

Por último, se puede buscar otro tratamiento de los procesos en el caso que compartan recurso, es decir, en este proyecto se mantiene el proceso interrumpido durante un tiempo estipulado. Este tiempo se podría hacer “inteligente”, es decir, que no se mantenga un tiempo, si no que se despierte cuando vea que el recurso está libre, evitando así algunos problemas de procesos que despiertan y reanudan antes de que el recurso quede libre. También se puede buscar otro tipo de tratamiento que no sea la espera como, por ejemplo, que realice otras acciones o parte del código que no altere al funcionamiento final mientras espera para poder acceder a ese recurso.

También, como se ha mencionado en las conclusiones de las pruebas, se puede realizar una modificación en el tiempo de espera de cada proceso, de manera que se duerma durante el tiempo de ejecución del proceso que está usando el recurso, pero no de manera íntegra, si no el tiempo que le queda al proceso para dejar el recurso libre. Esto se puede implementar mediante un ArrayList o una lista global que contenga todos los tiempos de ejecución de los procesos; y el tiempo que lleva ejecutándose el proceso que está ocupando el recurso.

Finalmente, aunque fuera de la funcionalidad global del proyecto, se podría mejorar tanto la interfaz gráfica de la herramienta, ofreciendo más funcionalidades como trazas, gráficas editables, mejor visualización de los contadores, etc.; como la comunicación cliente-servidor, buscando otro estándar de paso de mensajes, o reducir el tamaño de estos mensajes para que no lleve mucho tiempo el paso de mensajes de un cliente al servidor.

9. Glosario de términos

- ***Algoritmo:*** conjunto ordenado de operaciones sistemáticas que permite hallar la solución de un problema.
- ***API:*** interfaz de programación de aplicaciones, es un conjunto de subrutinas, funciones y procedimientos que ofrece una biblioteca a otro software para conseguir abstracción en la comunicación, representando así la capacidad de comunicación entre componentes de software.
- ***Arquitectura software***: conjunto de patrones que proporcionan un marco definido y claro para interactuar con el código fuente.
- ***Búfer:*** Espacio de memoria de almacenamiento temporal de información que permite transferir datos entre unidades funcionales.
- ***Bytecode***: código intermedio, más abstracto que el código máquina.
- ***Caché:*** memoria de acceso rápido de una computadora, que guarda datos recientemente procesados***.***
- ***Ciclo:*** período que tarda la CPU en ejecutar una instrucción de lenguaje máquina.
- ***Contador hardware:*** estadística o valor que se toma sobre un aspecto físico de la arquitectura.
- ***CPU:*** Unidad Central de Proceso, la cual interpreta las instrucciones de las que se componen los programas y procesa los datos.
- ***Cuello de botella:*** límite en la capacidad de transferencia de información de un sistema o una conexión, que puede reducir el tráfico en condiciones de sobrecarga. Produce un descenso tanto en el rendimiento como en la velocidad del sistema y de la conexión.
- ***Escalabilidad:*** propiedad deseable de un sistema que indica su habilidad para reaccionar y adaptarse sin perder calidad, o bien para ampliarse sin perder calidad en los servicios ofrecidos.
- ***Estado de suspensión:*** estado de parada de la ejecución de un proceso hasta que se recibe una señal de reanudación.
- ***Evento entrada/salida:*** eventos especiales en los que se accede a dispositivos que están compartidos como, por ejemplo, puertos o discos duros.
- ***Flag:*** bits que se utilizan para almacenar un valor binario que tiene asignado un significado. En ocasiones, este valor binario representará uno de los posibles estados del sistema.
- ***FLOPS:*** operaciones de coma flotante por segundo, indican la medida de rendimiento de una computadora.
- ***Hash:*** función computable mediante algoritmos que tienen como entrada un conjunto de elementos, usualmente cadenas, y los convierten en un rango de salida finitos mediante el mapeado.
- ***Herramienta:*** programas o aplicaciones usadas para efectuar otras tareas de un modo más sencillo.
- ***Host:*** computador conectado a una red, de la que se provee y utiliza los servicios que le aporta.
- ***Hilo:*** unidad de procesamiento mínima que puede ser planificada por un sistema operativo.
- ***Interfaz:*** dispositivo capaz de convertir las señales generadas por un sistema en señales comprensible por otro.
- ***Interrupción:*** señal recibida por el procesador para indicar que debe parar la ejecución actual y pasar a ejecutar código específico para tratar la situación.

- ***Kernel:*** o núcleo, es el software fundamental del sistema operativo, y se ejecuta en modo privilegiado.
- ***Librería:*** conjunto de implementaciones que ofrecen una amplificación de la funcionalidad del código. No son ejecutables, pero pueden ser usadas por programas ejecutables que las necesiten.
- ***Machine Learning:*** rama de la inteligencia artificial que desarrolla técnicas para que las computadoras sean capaces de *aprender*.
- ***Metalenguaje:*** lenguaje que se utiliza para hablar acerca de otro lenguaje.
- ***Minería de datos:*** proceso de detectar información procesable de un conjunto grande de datos, usando análisis matemático para deducir patrones y tendencias.
- ***Modelado predictivo:*** permiten estimar, mediante técnicas analíticas de minería de datos, el comportamiento esperado de un objetivo a partir de los datos recogidos de este.
- ***Monitorizar:*** controlar y observar el desarrollo de una acción o suceso a través de unos monitores.
- ***Multithreading:*** soporte hardware para ejecutar múltiples hilos de ejecución en paralelo.
- ***Mutex:*** algoritmo de exclusión mutua utilizado para evitar el ingreso a secciones críticas por más de un proceso a la vez.
- ***Parser:*** analizador sintáctico que permite el análisis de la jerarquía de un archivo de entrada, construyendo una estructura de datos que será manejable por un software determinado.
- ***Plataforma:*** gran software que sirve como base para ejecutar determinadas aplicaciones compatibles con este.
- ***Plugin:*** aplicación que se relaciona con otra para ampliar su funcionalidad específica, interactuando por medio de la API.
- ***Portabilidad:*** capacidad de ejecutarse en diversas plataformas.
- ***Procesador:*** componente electrónico integrado en un chip donde se realizan los procesos lógicos y funciones de los computadores electrónicos digitales.
- ***Puerto***: interfaz a través de la cual se pueden enviar y recibir los diferentes tipos de datos.
- ***Rendimiento:*** medida o cuantificación de la velocidad que tarda una tarea o proceso en realizarse o completar su objetivo.
- ***Rutina:*** subalgoritmo parte del algoritmo principal.
- ***Sistema Operativo:*** conjunto de órdenes y programas que componen el software básico de una computadora, proveyendo una interfaz entre el resto de programas del computador, los dispositivos hardware y el usuario.
- ***Socket:*** API para protocolos de internet, el cual constituye un mecanismo para la entrega de paquetes de datos de la tarjeta de red a los procesos, definido por un par de direcciones IP, protocolo de transporte y un par de puertos.

10. Referencias

Aldous, D. (1991). The Continuum Random Tree. I. *The Annals of Probability, 19*(1), 1-28.

Barcelona Supercomputing Center. (22 de 09 de 2016). *Paraver: a flexible performance analysis tool*. Obtenido de Paraver: a flexible performance analysis tool: http://www.bsc.es/computer-sciences/performance-tools/paraver

Gantt, H. (1910). *Gantt Chart History*. Obtenido de Gantt Chart History: http://www.gantt.com/index.htm

Gilbert, D. (22 de 09 de 2016). *JFreeChart*. Obtenido de JFreeChart: http://www.jfree.org/jfreechart/

GWT-TUD GmbH. (22 de 09 de 2016). *Overview: VAMPIR*. Obtenido de Overview: VAMPIR: https://www.vampir.eu/

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). *The WEKA Data Mining Software: An Update* (Vol. 11). SIGKDD Explorations.

ICL UR. (22 de 09 de 2016). *Overview: PAPI*. Obtenido de Overview: PAPI: http://icl.cs.utk.edu/papI/Overview/index.html

ICL UR. (22 de 09 de 2016). *Overview: PAPI*. Obtenido de Overview: PAPI: http://icl.cs.utk.edu/papI/Overview/index.html

Intel. (22 de 09 de 2016). *Intel Vtune Amplifier XE*. Obtenido de Intel Vtune Amplifier XE: http://software.intel.com/en-us/intel-vtune-amplifier-xe

Kelleher, J. D., Namee, B. M., & D'Arcy, A. (2015). *Fundamentals of Machine Learning for Predictive Data Analytics.* Mit Press Ltd .

LANS. (2007). *Jumpshot: Performance Visualization for Parallel Programs*. Obtenido de http://www.mcs.anl.gov/research/projects/perfvis/software/viewers/index.htm

Microsoft. (22 de 09 de 2016). *Windows 10: features*. Obtenido de Windows 10: features: https://www.microsoft.com/es-es/windows/features

MPICH Project. (22 de 09 de 2016). *Overview: MPICH.* Obtenido de Overview: MPICH.: http://www.mpich.org/about/overview/

Oracle. (22 de 09 de 2016). *About: Java*. Obtenido de About: Java: http://java.com/es/about/whatis_java.jsp

Shende, S., & Malony, A. D. (Summer 2006). TAU: The TAU Parallel Performance System. *International Journal of High Performance Computing Applications, 20*(2), 287-311. Obtenido de https://www.cs.uoregon.edu/research/tau/docs.php

Sublime. (22 de 09 de 2016). *Sublime Text 3*. Obtenido de Sublime Text 3: https://www.sublimetext.com/

The Eclipse Foundation. (22 de 09 de 2016). *The Eclipse Foundation*. Obtenido de Eclipse: https://eclipse.org

The GNOME Project. (22 de 09 de 2016). *Gedit: The GNOME Project*. Obtenido de The GNOME Project: https://wiki.gnome.org/Apps/Gedit

The GNOME Project. (22 de 09 de 2016). *The XML C parser and toolkit of Gnome: libxml*. Obtenido de libxml site: http://xmlsoft.org/

The Linux foundation. (22 de 09 de 2016). *What is Linux?* Obtenido de What is Linux?: https://www.linux.com/what-is-linux

Wikimedia foundation. (22 de 09 de 2016). *Perf's wiki*. Obtenido de Perf's wiki: https://perf.wiki.kernel.org/index.php/Main_Page

Wikimedia foundation. (22 de 09 de 2016). *Wikipedia: C (lenguaje de programación)*. Obtenido de Wikipedia: C (lenguaje de programación): https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)

Anexo A: Manual de Usuario

Introducción

El manual de usuario es una guía fundamental para el correcto uso de nuestra herramienta. En él, especificaremos los requisitos mínimos del sistema, tanto hardware como software para su correcto funcionamiento, así como las instrucciones para configurar y ejecutar la herramienta.

Requisitos mínimos

En este apartado se especificarán los requisitos, tanto hardware como software, mínimos para la correcta ejecución de la herramienta. Respecto al hardware, se resumen en los requisitos mínimos para usar Java y de memoria para albergar las aplicaciones creadas:

- Memoria: 128 Mb.
- Disco duro: 3Mb.
- Procesador: mínimo Pentium 2 a 266MHz.

Respecto a los requisitos mínimos software, estos se resumen en las librerías y lenguajes de la aplicación, además de algunas herramientas para la edición del XML. Estos requisitos son:

- ***Librería MPICH*** *(ha de estar instalada en el sistema).*
- ***Librería PAPI*** *(ha de estar instalada en el sistema).*
- ***Libxml2*** *(ha de estar instalada en el sistema).*
- ***Weka 3*** *(el ejecutable ha de estar situado en el directorio del servidor).*
- ***JFreeChart*** *(el ejecutable ha de estar situado en el directorio del servidor).*
- ***Notepad++ o Sublime Text 3 o Gedit*** para la edición del fichero XML.
- ***Entorno Java*** para la ejecución del servidor.
- ***Entorno C*** para la ejecución del cliente.

Configuración

El archivo XML es el utilizado como configuración para la medición de los contadores de la aplicación del cliente. Este debe de estar en el mismo directorio de la librería MPI modificada, y de la herramienta; y se compondrá del siguiente contenido:

```
<Configuración>
	<Conexión> //Variables para la conexión con el servidor
		<Establecer>0</establecer> //Realizar conexión. 0: No 1: Si
		<Puerto>6000</puerto> //Puerto estándar
		<Host>localhost</host> //Host donde se conectará al servidor
	</Conexión>
	<PAPI> //Variables que representan los eventos PAPI
		<Numero>2</numero> //Numero de eventos a medir
		<Evento>PAPI_TOT_INS</evento> //Nombre de un evento hardware
		<Evento>PAPI_FP_OPS</evento>  //Evento hardware que indicará eventos
                                      E/S (No modificable)
	</PAPI>
	<Algoritmo> //Variables dedicadas al algoritmo
		<Búfer>10000000</búfer> //Tamaño máximo del periodo
		<Reinicio>20</reinicio> //Reiniciar el algoritmo
		<Llamadas>1000</llamadas> //Número de llamadas mínimas
		<Sleep>1</sleep> //Retraso del hilo. 0: No 1: Si
	</Algoritmo>
	<Búfer> //Memoria para el hilo
		<Longitud>10000000</longitud> //Tamaño del búfer
		<Matriz>200000</matriz> //Tamaño máximo de la matriz
	 </Búfer>
	<Depuración> //Depuración de la herramienta
		<debugConexion>0</debugConexion>
		<debugHilo>0</debugHilo>
		<debugInicio>0</debugInicio>
		<debugFinal>0</debugFinal>
		<debugPAPI>0</debugPAPI>
		<debugHash>0</debugHash>
		<debugLlamada>0</debugLlamada>
		<debugAlgoritmo>0</debugAlgoritmo>
		<debugMatriz>0</debugMatriz>
		<Proceso>0</proceso>
	</Depuración>
</Configuración>
```

Compilación

Antes de ejecutar la herramienta, es necesaria la compilación, tanto de la aplicación a analizar con las librerías, entre ellas la librería MPI modificada; como el servidor con las librerías que utiliza.

Cliente

Para compilar el cliente se ha de utilizar el siguiente comando, especificado de forma general:

$$rutaMPI/bin/mpicc\ -I/rutaLibxml\ -lxml2\ -I/rutaPAPI/src/\ -L/rutaPAPI/src\ -lpapi\ -L/rutaPAPI/src/libpfm4/lib\ -lpfm\ MPI.c\ programa.c\ -o\ programa$$

Los parámetros a modificar son los siguientes:

- ***RutaMPI***: ruta en la que se encuentre instalada la librería MPI original.
- ***RutaLibxml***: ruta en la que se encuentre instalada la librería Libxml.
- ***RutaPAPI***: ruta en la que se encuentre instalada la librería PAPI.
- ***Programa***: programa a monitorizar.

Servidor

Para compilar el servidor se ha de utilizar el siguiente comando, especificado de forma general:

$$Javac\ -cp\ '.:jcommon-1.0.17.jar:jfreechart-1.0.14.jar:miglayout15-swing.jar:weka.jar:weka-src.jar'\ ServidorHilos.java$$

La aplicación del servidor, normalmente, se aportará compilada. Se especifica esta compilación en el caso de futuras modificaciones, si se quieren ampliar las funcionalidades, o parte del código.

Ejecución

La ejecución que ha de realizar el cliente es la forma estipulada por MPI, es decir, ha de referenciar al ejecutable de MPI, además del programa a monitorizar:

$$/rutaMPI/bin/mpiexec\ -np\ NumeroDeHilos\ ./programa$$

En el caso de la ejecución del servidor, es muy similar a su compilación, es decir, se han de referenciar todas las librerías que utiliza, con el siguiente comando:

$$Java\ -cp\ '.:jcommon-1.0.17.jar:jfreechart-1.0.14.jar:miglayout15-swing.jar:weka.jar:weka-src.jar'\ ServidorHilos$$

Anexo B: Abstract

Introduction and objectives

In this chapter, a introduction of the final grade work is made, including the motivation of its realization, the objectives –principal and secondary- of the project, and the structure of the project.

The motivation of this project can be summary in two words: improve performance. Nowadays, the performance is a important factor in the market and investigation scope. The use of supercomputer and clusters is very common in this scopes, so parallel computing is esencial for this systems. That is because this machines execute a very, very high computational cost programs, with a huge quantity of instructions, and the use of multiple nodes, process or threads is neccesary for a faster execution.

This machines can be as big as the companies or scientist want, but it will always be finite. This causes a huge problem: if it is finite, its resources will be, so it has a limit; and when threads or process in the same machine reach that limit, the machine starts to share the use of its resources, causing performance or even information leaks.

Most of this performance leaks reside in when the machine makes I/O instructions when it executes applications. These instructions communicate, through buses, with the CPU. Is in this access to these CPUs where the performance problem resides, because two or more threads or process could need to access to the same CPU at the same time, causing bottleneck problems, and this will adversely affect to performance and execution time of the applications.

This performance leak problem will be treated in this project, through I/O events prediction, using a technique that is currently booming, Machine Learning, i.e, using artificial intelligence for learning the I/O patterns and, from this learning, can deduce, or predict, future events.

This is part of the main objective of the project. This objective is the modification of a performance modelation and evaluation of applications; with the goal of search, locate and predict I/O instructions in the applications evaluated; and decide what action make to avoid collisions and resource sharing. For that, from the values received from the hardware counters made by the application, the counters that show the I/O instructions will be monitorized, and with the goal to predict the next ones, a Machine Learning model will be done.

The Project also has especific goals, detailing and complementing the main objective:

- Improving the communication between server and client. The actual communication is one way: client → server. Our Project needs to send information from server to client, to send actions against bottlenecks in resource sharing.

- Learn and choice a Machine Learning algorithm who has the better measure for our application. Machine Learning owns a lot of data models and we have to find the better algorithm for our problem and the data that the program collect.

- Show to the user the information analized in real time. For user interaction, the tool will show the values of the hardware counters that it collect.

- Finding a pattern in the counters that shows I/O instructions. This pattern – either instantaneous increase or decrease of counter values – will help us to see if the Application is making I/O instructions.

- Collect data for Machine Learning. The program will collect data obtained in every execution cicle, for provide the Machine Learning model of knowledge that will increase the model and prediction.

- Reduce the time wasted on sharing CPU between applications, through induction of a waiting time to the aplications that will use a busy CPU.

Architecture summary

The tool architecture will have 2 main parts, a modiffied MPI library, written in C, which will have the objective of read the counters, send information to server, and make the Application sleep during the time received from the server; and this server, written in Java, with the objective of show the information received, detect and predict I/O events and decide what action make to avoid bottleneck.

Global

The application, once started, must realice a inicialization call to the modiffied MPI library, which will be called continously during the code execution of the application; and, at the end, a finishing call must be sent to the library. This moddified library will capture all the calls to the original MPI library, and will collect them before redirect them to the original library.

The server will be executing in a parallel way, independently. This server will be waiting for a conection until it receive a conection petition from a client. In this moment, the server will create a thread that will treat the information reception from the library, to be shown and analized by the interface.

Library

As we said, the funcionality of the library is capture the events, send them to the server and the threatment of the clients in execution, depending of the information received from the server. For this, first PAPI must be inicialized, creating the EventSet – where the PAPI events are saved – and loading the configuration of the library.

This configuration is saved in a file, which contains:

- Conexión: parameters for the conection, port and host.
- PAPI: this parameter let you choose the number or type of events or counters to read. In our Project, one of this events must be PAPI_FPS_OPS. This counter react with the I/O events, because when an I/O instruction happen, the machine doesn't use the FLOPS operations.
- Algoritmo: parameters useful for period detection, as well as for the restart of the algorithm or the delay of the thread execution.
- Búfer: parameters that shows the buffer width saved, and the matrix sent to the server.

After read the configuration, a original MPI call will be made, inicializating this library. Then, the server create a thread, that will start a connection with the server. This conection will be stablished by sending a petition to the default port, and the host indicated in the configuration file. Then, the library receive a port, and make the connection with this port. After this, the execution of the library starts. This execution can be divided in two phases:

- Data processing: this is the original funcionality of the tool. It will check if there is data to read. If there is not data, it will wait to have it. Else, it will ask for the mutex to access to the buffer of the calls, and then this data will be loaded then and be sent to the period detection algorithm. Then, the mutex will be set free and this matrix will be send to the server.

- Result treatment: this phases will start just right after sending data to the server. The server will send to the library a Message with the treatment to take, in this project it will be the time that the process will be sleeping. When the process restart to execute, it will a normal execution.

The rest of the funcionalities will be the same as the original tool: the capture of the calls will be done by capturing MPI calls, and the calculation of the time and counters before and after the MPI call; and it will be saved in the buffer, with its hash; and the finalization of the library will be done with the capturation of the MPI finish call, redirected to the original MPI library and destroying the mutex and the thread, stopping also the PAPI counters.

Client – Server Connection

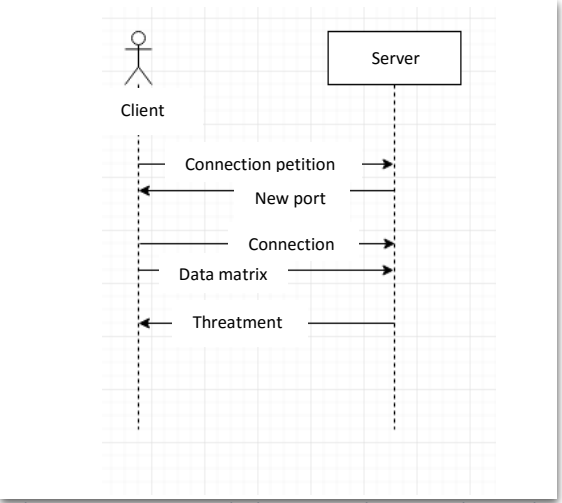

Ilustración 38. Diagrama de la conexión actual – ingles.

The tool will have a two-ways connection: the library send the counter info and receive the waiting time; and the server receive the matrix with the info and send this waiting time.

Server

The server changed its function in the tool. Now it is an active part of the system, detecting and predicting I/O events. The first step is inicializate it, opening the configuration of the server – with the same attributes as the library – and creating a socket in the specified port; starting the execution of the server. First of all, the WEKA library starts, and the connection is made in the same way as the library. Then, the server starts to receive the data, being shown in the interface. Then, starts the I/O detection and prediction.

The counter target for this actions is PAPI_FP_OPS. If there is an I/O event, this counter will decrease or increase a huge amount of operations. If it happens there is 3 ways to resolve it:

- If the server detected I/O events:
  - If there is no other thread using the resource, the server change the flag value to occupied, accessed by a mutex, to avoid coherence problems.
  - If there is another thread using the resource, the number of threads waiting is increased by one, and the waiting time is calculated, sent to the client. The number of threads waiting is a global variable too.
- If there is no I/O events, the execution is normal.

After this checking, the server will save the values of the counter, the time since the last I/O event and the application status in an .arff file, a WEKA database format. This will be used by the Machine Learning library, creating a Random Tree model from this data. After this, from this model it will predict the next value with a gap of 2 cycles.

Testing summary

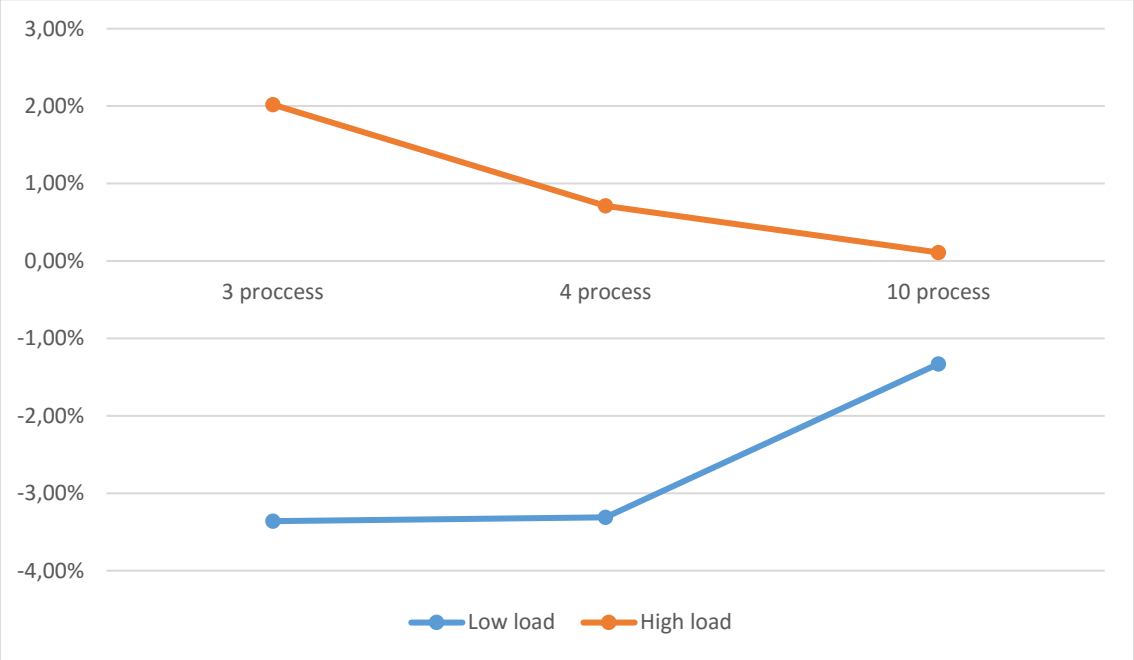

Ilustración 39. Gráfica de resultados – inglés.

As we can see, there is an antithesis depending on the load being assigned to the I/O of the thread. When we have applications that perform low loads, as the range of 1,000 as block size, we obtain a higher performance benefit when more processes are executing at the time.

Instead, when we have applications that perform high loads, as the range of 10,000 as block size, we can see that the performance benefit is degraded as we add processes in parallel.

The reason for these results could reside in the time that process wait, sent by the server. This was implemented so the time was set based on the time between the I/O events, being the same time in all implementations, without depending of the number of iterations in the executions. This implementation has been found to not provide the expected benefit to the system by the following:

- Low load: the time between I/O events is greater than the time it takes to run these events; which carries a wasted time at the end of each run. This can be reflected in the fact that the execution of three processes have a worse performance benefit than 10 processes because run 3 processes coinciding in parallel entails less time than the 10 processes, therefore the extra time wasted by this waiting it is greater in the case of less processes.
- High Load: this case is just the opposite. The time between I/O events is less than the duration of the execution of these events, seeing that reflected in graphic, having a completely opposite behaviour to the previous case. This leads a much better performance in the case of three prcesess, because the proportion of time covered by the sleep time relative to the total execution time is greater than in the case of 10

process; because this one takes longer to run and, as we said before, the waiting time is the same in both cases (200 iterations).

Seeing this, the ideal case would be to fix it by changing this waiting time. This time, if it is replaced by the time it takes to I/O events to run, the system would obtain an better performance improvement. We proceeded to implement a mode in which the waiting time will be the running time of the events, always keeping the shortest of time. However, the results were not the expected.

This is because this time remains constant whether the bottleneck happens at the beginning or the end of the execution of the process that is using the resource, obtaining a waste of time, which leads to higher performance degradation in the case where the waiting is performed at the end of the execution of this process.

In conclusion, the tests made with the two cases have not proved to be the expected for this system. In the first case, it offered a few benefits for large blocks of data; while in the second the penalty has been global. As a possible solution to these problems with waiting time, proposed a dynamic time, not only consider the execution time of each process, but also the percentage or portion of that time that has already been executed when processing a request access to that resource, minimizing the most of the wasted time when the resource is free.

Conclusion

This proyect has as main goal expand the funcionality of the application monitoring tool to avoid the performance leaks made by resource sharing, in this case, I/O events; improving the performance by predicting this events and threating the process from this prediction.

The first step taken was to make a study and understanding of the tool and its code; in addition to make a research on similar tools on the market, and a study of the features offered by these. From this step the specific objectives of the project were stipulated and, of them, libraries and minimum system requirements were analyzed.

The next step was to choose in which area of the original code I will implement our new functionality. First, I must find the method where the client-server communication was made and, after studying the MPI protocol, a two-way communication was implemented, improving the functionality of the communications system and allowing the server could communicate with the client; reaching the goal of the objective of improving the communications system.

The next objective was the effectively detection of I/O events. First, the behavior of the I/O instructions and machine components used during the execution were analyzed. It was concluded that when an I/O instruction was made, no FLOPS operations were made. Finally we had to implement a reading of those instructions in our tool. This was possible thanks to the original monitoring tool that allowed us to verify that a FLOPS operation counter, included in the PAPI library, had a pattern of decrease when these I/O instructions were executed; fulfilling the objective.

Then, I searched for a statistical method to predict when one of these events would happen. For this, many technologies were studied, reaching the conclusion that the most suitable was the Machine Learning, from the WEKA library, which was to be used for this prediction. But the WEKA library offered many models of prediction data, so I had to perform an experiment from the data collected by the original tool, to see which was the model for a more efficient and accurate prediction in our case. The solution obtained was a Random Tree model because it offered less absolute error when it predicts an event, reaching the goal of the objective.

But this prediction model needs a data set in order to perform correctly. For this, the WEKA library has a file format, arff, for storage this data; and a catalog of functions for creating, reading, writing and interact with files of that type of format. Thanks to this great functionality of the WEKA library is possible compliance with the objective of collecting data for the use of Machine Learning.

Once predicted these events, the choice of a treatment was necessary to try to improve the performance in the case of bottleneck and resource sharing. The easiest way was chosen: if several processes access at the same time to a resource, and then there is a long period without any use of that resource, so the processes that want to access the resource occupied can be moved to those periods which is unoccupied. This was possible by implementing a wait that keeps the process in suspension until a time chosen, reaching the goal.

Finally, all data collected from these counters are shown to users through the functionality of the base interface tool, so that the target of showing real time information is reached.

All the compliances of all specific objectives also ensures compliance of the main objective: to create a tool that predict I/O events and tries to improve their performance; although the final results, as discussed in the section, are not the expected because of the erroneous timeout decision.

The experiences gained with this project, plus personal views on the subject and the tool to finish, will be discussed here:

- I was allowed to have a higher awareness of the importance of performance for the creation and implementation of tools, both when choosing tool, such as offering them; because it is a factor taken into account today.
- This also allowed me to observe how fragile is this performance because many factors, of code, the machine chosen, etc., will affect it.
- Also, a project of this magnitude has offered me a challenge that made me evolve as a student, as it is one of the first occasions that the opportunity of creating a project so important was given to me.
- I have also learned to accept mistakes and setbacks; gaining maturity and organization in solving these problems.

- I have adquire, finally, a better understanding of all subjecys of Computer Engineering and a very interesting technologies and libraries, important for future projects.

Finally, the cost of this project reach the amount of **17.997,41€**, taxes included.

Future plan

This project is focused only in one problem of performance: limited resource sharing and, inside this problem, only in I/O instructions. In future projects this tool funcionality could be extended, reaching a wider range of problems of performance, improving it and getting a more favorable results.

Another future project could be based on WEKA library. Machine Learning techonology scope is in constant growth and, each short time, a new technology or a improved version can be found. Probably, within months or years, a new prediction data technology will appear, more efficient than the one used in this project, and a better and more effective prediction can be obtained, without the need of a big retard on the prediction, as in this project, improving the performance of the tool too.

The read and detection of the events can be improved too. In this project the FLOPS counter is used, reading the FLOPS that the application uses in its execution; but there is a library, extension of PAPI library, called appio, which has a very interesting tools for read counters I/O. This library could offer more clear results, offering the posibility of simplificate the code too, both server and modified MPI library; improving the execution time of both programs, and improving its performance too.

Ultimately, you can search for another process treatment in case of resource sharing, i.e., in this project the process remains interrupted during a stipulated time. But this time could be “intelligent”, i.e, that doesn’t remain a time, however the process will wake up when the resource will be free, avoiding some problems, like a process that woke up when the resource is still ocuppied. Another method of treatment can be searched too. For example, instead of interrupting the process, it could execute part of code or another actions that don’t modify the behaviour of the application while is waiting for use the resource.

Finally, out of the global funcionality of the project, the graphic interface of the tool could be improved, offering more funcionalities, such as traces, personalizable Graphics, a better counter visualization, etc; as well as the client-server communication, looking for another Message passing standard, or reducing the size of this messages for reduce the time that Message Passing takes.